1

US ARMY
LABORATORY COMMAND

AD-A231 694

CONTRACT REPORT NUMBER 2-90
PREPARED FOR THE

HUMAN ENGINEERING LABORATORY

BY

Advanced Decision Systems
1500 Plymouth Street
Mountain View, California 94043-1230

August 1989

INTERACTIVE MODEL BASED VISION SYSTEM FOR TELEROBOTIC VEHICLES

Contract No. DAA15-88-C-0054
(ADS TR-3221-01)

US ARMY HUMAN ENGINEERING LABORATORY ABERDEEN PROVING GROUND, MARYLAND

HEL

ABERDEEN PROVING GROUND, MARYLAND 21005-5001

ADVANCED DECISION SYSTEMS
1500 Plymouth Street, Mountain View, California 94043-1230 • (415) 960-7300 • FAX (415) 960-7500

ADS TR-3221-01

Interactive Model Based Vision System for Telerobotic Vehicles

Daryl T. Lawton
Marcel Schoppers
E. David DiSabatino

August 1989

FINAL TECHNICAL REPORT
Period Covered: September 16, 1988 to March 15, 1989
Contract No. DAA15-88-C-0054
ADS Project No. 3221

Prepared for: U.S. Army AMCCOM
Aberdeen Proving Ground, MD
21010-5423

U.S. Army Laboratory Command
Human Engineering Laboratory
Aberdeen Proving Ground, MD
21005-5001

'NCLASSIFIED

JRITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS None
SECURITY CLASSIFICATION AUTHORITY None DECLASSIFICATION / DOWNGRADING SCHEDULE None	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
RFORMING ORGANIZATION REPORT NUMBER(S) ADS TR-3221-04	5 MONITORING ORGANIZATION REPORT NUMBER(S)

IAME OF PERFORMING ORGANIZATION vanced Decision Systems	6b OFFICE SYMBOL *(If applicable)*	7a NAME OF MONITORING ORGANIZATION US Army AMCCOM
DDRESS *(City, State, and ZIP Code)* 1500 Plymouth Street Mountain View, CA 94043-1230		7b ADDRESS *(City, State, and ZIP Code)* Aberdeen Proving Grounds, MD 21010-5423
IAME OF FUNDING / SPONSORING)RGANIZATION US Army Laboratory Command	8b OFFICE SYMBOL *(If applicable)* SLCHE-CS	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER DAAA15-88-C-0054

DDRESS *(City, State, and ZIP Code)*
Human Engineering Laboratory
Aberdeen Proving Ground, MD 21005-5001

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO

ITLE *(Include Security Classification)*
teractive Model Based Vision System for Telerobotic Vehicles (Unclassified)

ERSONAL AUTHOR(S)
Daryl T. Lawton, Marcel Schoppers, and E. David DiSabatino

TYPE OF REPORT	13b TIME COVERED	14. DATE OF REPORT *(Year, Month, Day)*	15 PAGE COUNT
l Technical (draft)	FROM 9-16-88 TO 3-15-89	89SEP27	119

JPPLEMENTARY NOTATION

COSATI CODES			18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
IELD	GROUP	SUB-GROUP	
17	07		
17	08		

3STRACT *(Continue on reverse if necessary and identify by block number)*

:tempts to develop intelligent and completely autonomous systems for battlefield applications have been largely unsuccessful te, in spite of continued progress in the underlying technologies. There remain unresolved and fundamental difficulties in of the necessary computational power, the required complexity of perceptual systems which can operate in complex, natural, ften hostile environments, and the corresponding complexity of planning and reasoning systems. Our approach to these :ms has been to develop a model based vision system that a human controls interactively. The human uses this system to y interpret sensory information from a distributed team of telerobots. The resulting interpretation will be a model of the that the telerobots can refine autonomously and also use to control their behavior. In this way the human can direct the bots by initializing processing that can then be handled autonomously. In addition, the communication between the robot ıe human will take place in the context of a shared model of the world. This report presents our design of such a system ıe development and initial testing of many of it's critical components regarding object representation, constraint-based ıce, terrain models, perceptual processing, and a user interface.

STRIBUTION / AVAILABILITY OF ABSTRACT JNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified	
AME OF RESPONSIBLE INDIVIDUAL Charles Shoemaker	22b TELEPHONE *(Include Area Code)*	22c OFFICE SYMBOL SLCHE-CS

RM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Limited Automated Perceptual Processing Capabilities - Attempts to deal with the control of multiple vehicles by increasing the autonomy of individual vehicles will have to confront the fact that autonomous perceptual systems which can reliably interpret such complex, outdoor imagery do not yet exist. These problems are exacerbated when dealing with passive data sensors and limited a prior information, as would be the case in a rapidly changing environment which is characteristic of a battlefield.

Our approach to these problems has been to develop a model based vision system that a human controls interactively. The human uses this system to rapidly interpret sensory information from a distributed team of telerobots. The resulting interpretation will be a model of the world that the telerobots can refine autonomously and also use to control their behavior. In this way the human can direct the telerobots by initializing processing that can then be handled autonomously. In addition, the communication between the robot and the human will take place in the context of a shared model of the world. The top of Figure 1-1 shows the direct connection between a human and a robot using a telerobotic link. In this the human is controlling a single vehicle. The bottom of Figure 1-1 shows the framework towards which we are building. It shows multiple telerobots with limited amounts of autonomy interacting with the human controller through a common model of the world.

1.1 Phase I Achievements

In Phase I of this project, we developed the overall architecture of such a model based vision system and developed and implemented some of its key components. The overall architecture is shown in Figure 1-2 and is described more completely in Section 2 of this report. Figure 1-3 shows the use of some of the initial components of the system. The human is interactively placing a ground plane into a model of the world and having it back-projected onto an image. Some surfaces have been placed in the model of the world and slid along the ground plane. These correspond to potential obstacles. In the complete system, the telerobot would verify and refine the placement of these objects as well as instantiate other objects into the world model.

A significant amount of work was performed during Phase I work to define and realize this system. In particular, we:

- Designed the overall system architecture,
- Implemented an initial version of the interactive interface in a Mac II,
- Developed a format for describing terrestrial object models which can be interactively instantiated by a human to form an interpretation of a scene,
- Investigated the underlying constraint manipulation and propagation mechanisms to assure consistency of relationships in the established world model,

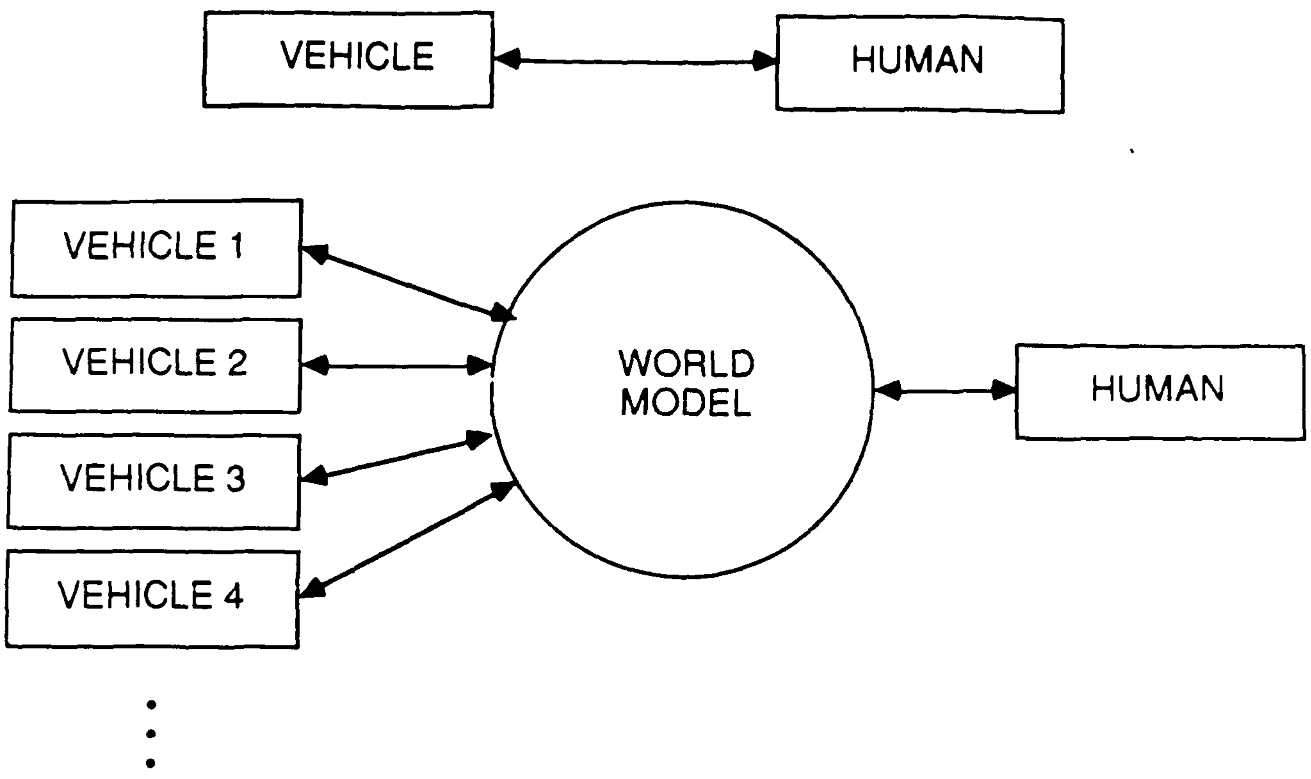


Figure 1-1: Use of Common WORLD Model for Control of Multiple Telerobots

- Developed an interface to autonomous perceptual processing during instantiation of object models, and
- Specified sensors for the telerobotic drone to autonomously refine a scene model.

1.2 Outline of Report

Section 1 of this report is an introduction that describes the key problems being addressed and our general approach to solving them.

In Section 2 we present the system architecture for an interactive model based vision system that a human can use to provide initial environmental interpretations for a telerobot. Critical components of this architecture are addressed in later separate sections.

Section 3 reviews work in object representation and processing using constraints. This is one of the fundamental technical problems in realizing our architecture because the models of world objects need to enforce realistic consistency when they

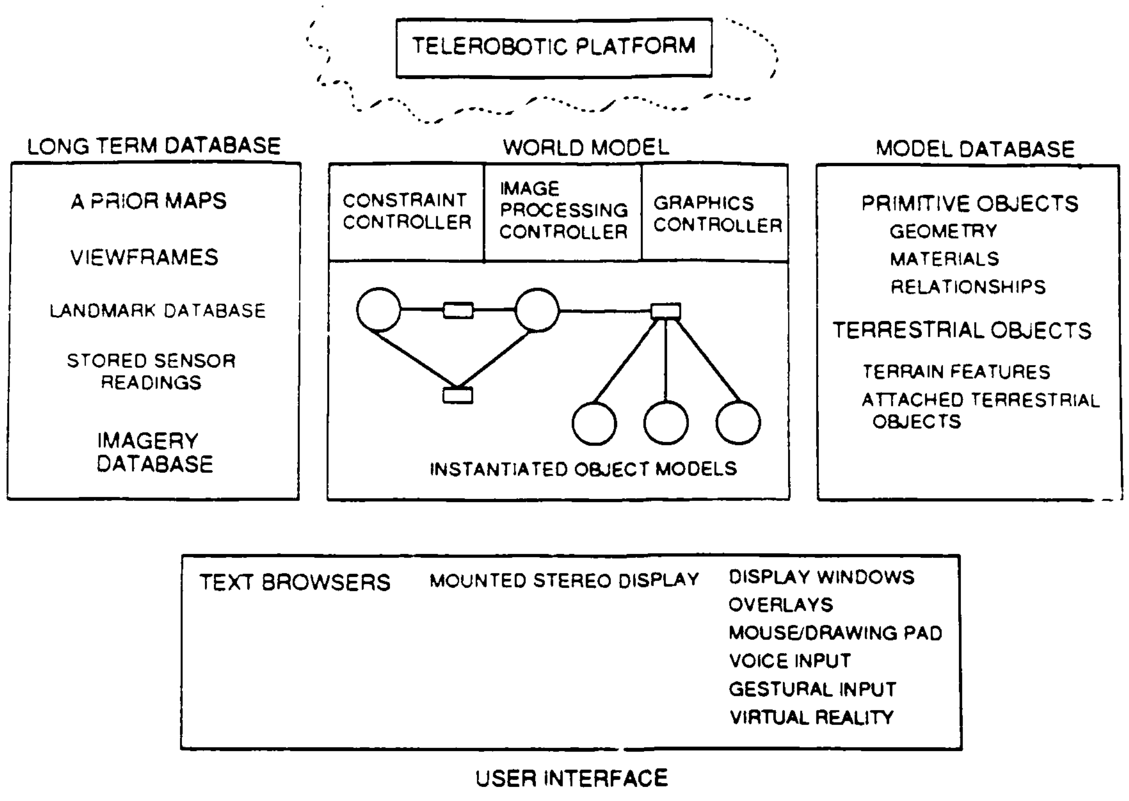


Figure 1-2: System Architecture

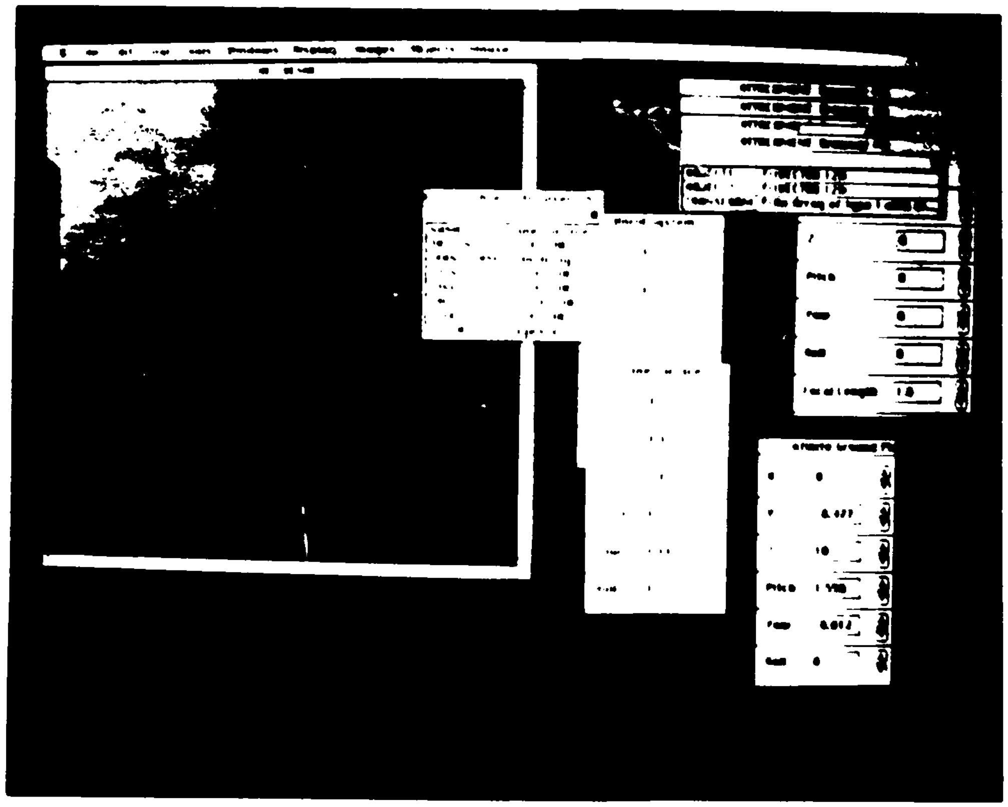


Figure 1-3: Prototype System Interface

are manipulated by the human and the telerobotic system. The human must use minimal actions to instantiate an object in the developing model of the world and the system will need to use the model to refine the interpretation. For example, the human may indicate that a tree is in a particular position relative to an image and the autonomous system will have to determine the location of the tree, where it's trunk is, and that it is in contact with the ground plane, and aligned with gravity using information associated with the general model of a tree. When the human manipulates the instantiation of this model, it must change in such away that the constraints which define the object are not violated.

Section 4 describes the general format of object models that we have developed to be realized using the general constraint processes techniques described in Section 3. We have developed a class of primitive objects which can be combined to model general objects in the environment and their inter-relations.

Section 5 describes work on the representation of perceptual information and the the types of processing which is done to extract more complicated relational information from sensors. The discussion is biased towards the segmentation of images into the types of patterns that are predicted from the objects that a human instantiates in the world model. Thus a human may roughly indicate that a terrain patch occurs in some area of an image and the system will use the material attributes of that type of terrain patch, along with the rough constraints provided by the human, to form a more exact area.

Section 6 describes one of the system databases for storing a prior map data along with previous interpretations of scenes obtained at different locations as the telerobots explores the environment.

Section 7 describes an initial user interface for the system we developed on a Mac II.

Section 8 describes our conclusion and plans for future work.

2. System Architecture

The underlying software architecture of our system (Figure 2-1) is built around three major data bases that a human can access and manipulate through an extensive user interface. The human will also be in contact with and can control the sensors onboard a distributed team of telerobots. The basic task of the human is to access general models of various types of terrestrial objects stored in the **Object Model Data Base** along with information in the **Long Term Data Base** which describes maps and previous interpretations of portions of the environment. Those are used to build an interpretation of the current scene in the **World Model Data Base**. In a simple version of this, the human is presented with an image taken from sensors on the telerobot. He can then use a prior maps to align landmarks and terrain features from these maps with the image. He can also access the three-dimensional and physically based models of objects and position them with respect to the world model. As he does this, the models are projected back against the images obtained from the telerobots. In the eventual realization of the system using head-mounted video screens, the human will have a sense of "reaching" behind the image and into the world itself as he constructs the model. The world model will describe enough features for the telerobot to function autonomously and also to communicate with the human in terms of a completed interpretation. The interpretation will in general take place with respect to a set of images obtained at multiple times and from multiple positions by different telerobots. More advanced user interfaces will allow the human to flip from view to view and to determine relationships between different views that can be incorporated into the world model.

2.1 Object Model Data Base

The Object Model Data Base contains generic models of objects, relationships, and events found in terrestrial scenes. This includes a wide range of objects such as terrain patches, vegetation, vehicles, trees, and gravity. There are two different types of objects: **Primitive Objects** which correspond to the basic entities and relationships used to describe and represent **Terrestrial Objects** which correspond to the more complex and conventional objects found in the world. Primitive Objects describe characteristics such as shape, material composition, and relationships between subparts. Terrestrial objects describe items such as roads and trees.

The representation and description of objects for an interactive vision system of the type outlined here, is significantly more complex, through related in many ways, to those used in CAD/CAM systems and geometric modeling packages because they will be manipulated by the system for autonomous processing. Thus, our model of a tree needs to also include that it is acted on by gravity and will have a preferred type of orientation and attachment with respect to the ground surface.

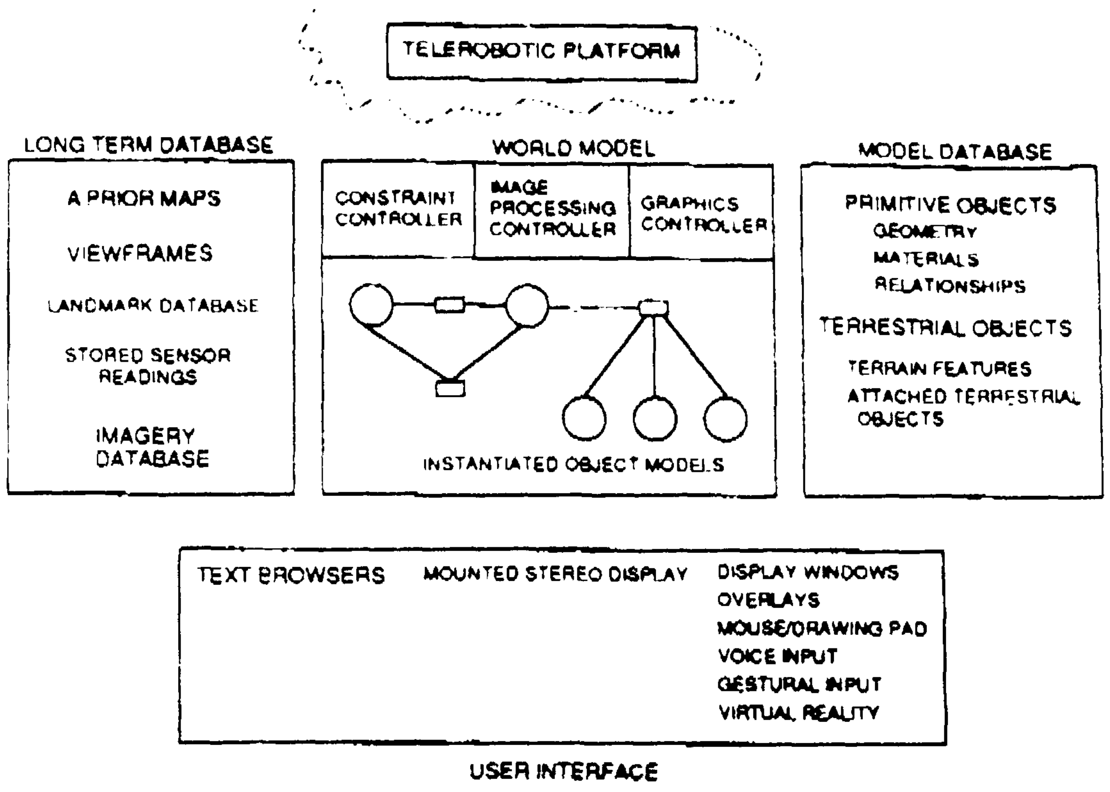


Figure 2-1: System Architecture

To achieve these, we use object-oriented programming techniques to describe world objects [Cox - 84]. This allows us to define objects as being made-up of other objects in a controlled manner. We can also associate programming methods with objects that are specialized for restricted classes of objects. Objects are defined using constraint-oriented descriptions [Borning - 79, Borning - 81, Mundy, Vrobel and Joynson - 89]. This adds to the flexibility of instantiating objects in the world model and helps free the human from having to specify an absurd amount of detail when interpreting a scene. When he changes one aspect of an object in a scene, the other related parts of the object change under the specified constraints. Many of the constraints between a sensor object and a particular type of object have attached procedures which correspond to perceptual actions.

2.2 World Model Data Base

The World Model Data Base is an active data base shared by the user and the telerobots. It describes the three dimensional world of objects and situations surrounding the telerobot. It is initially formed by the human accessing models in the model data base and instantiating them in the world model. The robot can then refine aspects of these instantiated objects such as their position and attributes. It can also begin instantiation of other objects based upon relationships stored in the model data base. There are three types of controllers associated with the world model data base. The **Constraint Controller** checks for consistency in the world model. Object models are described by sets of constraints which must be satisfied. A simple constraint is that the value of some parameter associated with an object model will be in some bounds (such as a tree being a particular height). More complicated constraints deal with relations such as a tree being connected to the ground or its relation to gravity and the orientation of the ground plane. Constraint processing can automatically ascertain values for an object instance. The human will in general specify a limited amount of information for an object and the system will use the constraints and processing actions associated with them to refine the model. The Constraint Controller can thus use the constraints which define an object or relationship to refine an instantiation or to find a violation of a constraint and ask the human for help. The **Perceptual Processing Controller** deals with the extraction of information from images and sensors on the robot. When the human indicates a particular object occurring at a given location, the constraints in the object model also specify the types of processing that needs to be done at the corresponding part of the image to obtain this information. There are active sensors on board the robot (or corresponding information determined by the intelligent use of passive sensors) that can obtain this information directly. Thus, when the human indicates that an object is somewhere, the corresponding image areas are marked out and the type of segmentation corresponding to the material class of the object is applied or the range sensor is used to refine its position. The **Graphic Controller** deals with the presentation of the world model and interactive scene measurements to the user. Thus when the user accesses a model of a tree, he is presented with a cartoonish 3D tree template which is back projected onto the image he is interpreting. The Graphic Controller does this presentation. It is also used to present the world model at a given stage of processing to the human.

The world model is the description of objects and events in the physical environment. It is not based on a particular sensor or time. It will in general contain multiple coordinate systems each associated with several different sensors. Each of these coordinate systems can be conditionally linked together by an explicit rigid body or qualitative transformation. Thus something can appear and be represented in one sensor but may not be described in the coordinate system of another sensor.

When the user completes the instantiation of the object model, it is refined by the system in three different ways: 1) it checks for constraint violations with respect to the newly instantiated object and other objects in the world model; 2) it will modify unspecified attributes based upon the constraint relationships in the object model; 3) it will begin autonomous processing to instantiate other objects and relationships based upon the specified constraints. Thus, when an terrestrial object is instantiated, if there is no ground plane, one will be instantiated using the corresponding points on any terrestrial objects which have been placed in the world model. Or it will direct an active sensor (such as a laser or a chemical probe) or use passive sensor processing (such as stereo) to refine the attributes of the instantiated object (such as how far away it is, or what its material properties are). This can involve calling an image processing or active sensing routine to do such things as fitting the contour of the object or to applying histogram guided segmentation based upon the expected material attributes of the object.

The human is ultimately responsible for directing the formation of a consistent model of the world. The telerobots are responsible for filling this model in and notifying the human of discrepancies. The world model can be shared and accessed by multiple telerobots. This will be the basis by which a single human can control multiple, spatially distributed telerobots.

2.3 Long Term Data Base

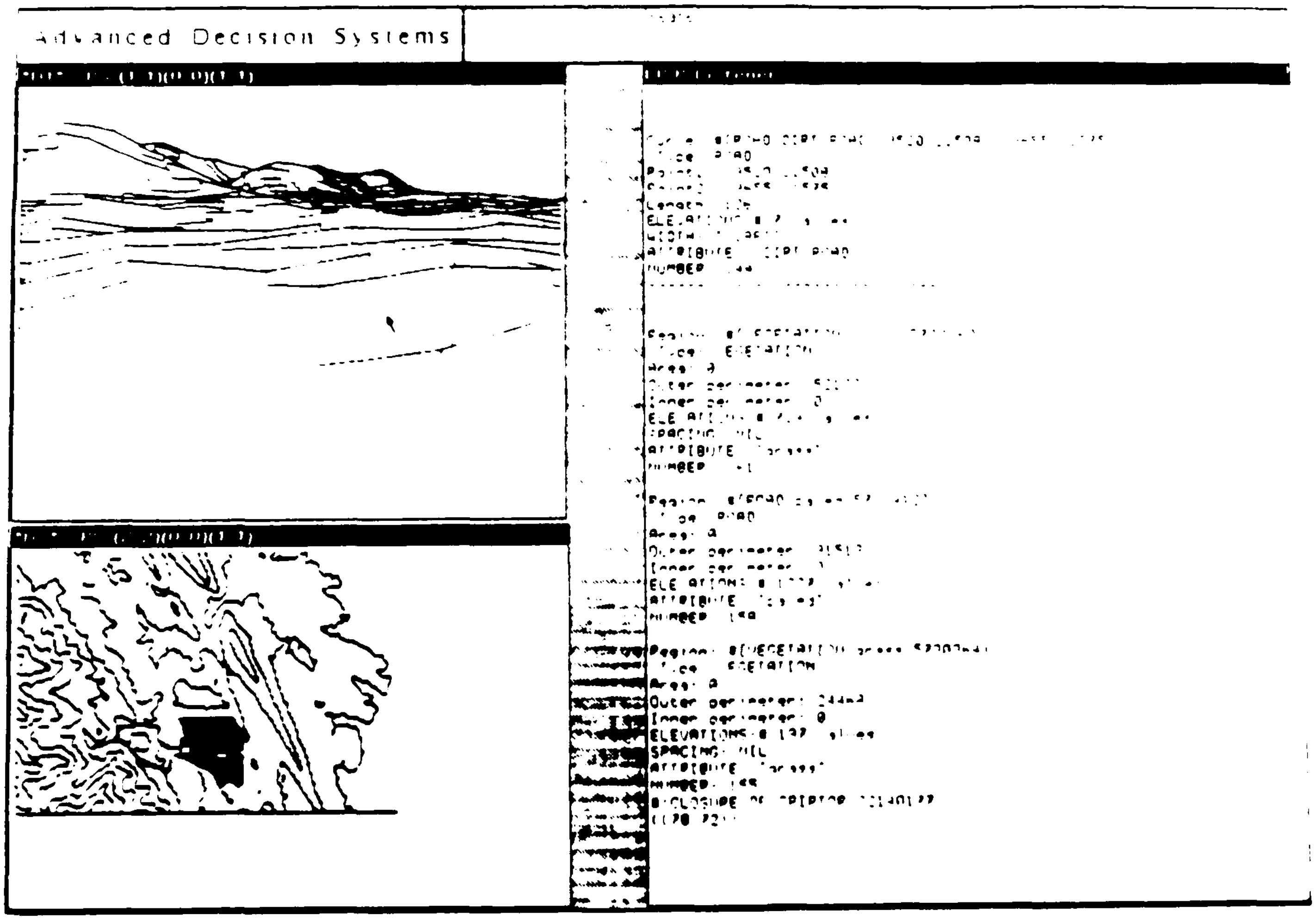
The Long Term Data Base contains a prior information such as terrain maps; imagery and sensor readings obtained by the telerobot over time; and scene models established by the telerobot at other locations in the environment. These previous interpretations can come from either the robot's past explorations or by other telerobots operating roughly at the same time.

2.4 User Interface

The User Interface consists of mechanisms to access and move information between the different data bases. The human also depends on the interface to have information presented to him regarding the current state of the world model.

A key attribute of the user interface to the world model is what we term a label-plane depth buffer (Figure 2-2). This is a conventional depth buffer used for hidden surface removal in computer graphics [Foley and Van Dam - 82] which is modified to have a set of pointers, ordered by depth, to all the objects in the world model that project onto a given pixel in the image. Thus when the human "touches" a pixel in the image from the telerobot, he can access all the objects in the world model that project onto that pixel. In the figure, all the terrain classes which occur within the given image area are returned for the area the mouse-arrow is pointing to.

The human can place objects into the world model in several different ways. In one form he can access the objects and manipulate them via their three dimensional attributes with respect to some coordinate system associated with the world model. When it has been positioned as desired, the different components of the object can be placed into the label-plane depth buffer associated with the image. In this way, the projected attributes of the instantiated object can access the actual image or the results of image processing routines. The user can **Burn** attributes in when he instantiates an object. Burning means that the attributes can not be changed. In general, this means constraining a particular feature to lay along a ray of projection. Another technique is for the user to directly draw the specified object on the sensory



```
Curve [illegible] DIRT ROAD [illegible]
 Type ROAD
Point1 [illegible]
Point2 [illegible]
Length [illegible]
ELEVATIONS [illegible]
WIDTH [illegible]
ATTRIBUTE DIRT ROAD
NUMBER [illegible]

Region [illegible]
 Type VEGETATION
Area [illegible]
Outer perimeter [illegible]
Inner perimeter 0
ELEVATIONS [illegible]
SPACING NIL
ATTRIBUTE [illegible]
NUMBER [illegible]

Region [illegible]
 Type ROAD
Area [illegible]
Outer perimeter [illegible]
Inner perimeter [illegible]
ELEVATIONS [illegible]
ATTRIBUTE [illegible]
NUMBER [illegible]

Region: #(VEGETATION [illegible]
 Type VEGETATION
Area [illegible]
Outer perimeter: [illegible]
Inner perimeter: 0
ELEVATIONS [illegible]
SPACING: NIL
ATTRIBUTE [illegible]
NUMBER: [illegible]
#<CLOSURE [illegible]
((70 72 [illegible]
```

Figure 2-2: Label-Plane Depth Buffer

input and then indicate the attributes of the object which has been drawn. As example of this is interactively segmenting an image into different types of terrain patches and pointing out that different edges correspond to terrain feature discontinuities. The object model is still accessed by the user in this case, but the user specifies that he is directly burning in an attribute: This means that it is then set and not subject to alteration. When the user draws on the image, he is specifying a set of incidence constraint for the extracted features. Thus, the system can move the model with respect to depth in the world model but in no other way.

The eventual system will be defined by a sequence of user interfaces. Initially, the user interface is based upon widows for displaying imagery and graphical overlays and text-based browsers for inspecting entities in the data bases in detail. This basic level of interface can be quite tedious to work with and it's future role will be to serve as a debugging tool for the later development. An intermediate system interface will use a more natural set of tools such as 3D tactile position sensors and voice activated control. Using these, the user will actually have a sense of reaching into a data base of models and then placing them in the world model. In the finally realized system, the world model and the sensor input from the different telerobots is presented to the human as a **virtual reality**. In this, the human can receive direct sensory input either from one of the telerobots or the world model itself.

2.5 Telerobotic Sensor Platform

Even though the actual construction of the sensor platform used by the telerobot is not within the scope of the work proposed here, we are able to describe several desirable features it should have. The onboard sensors need to be controllable both by the human interactively using the system and also automatically by the telerobot itself when it instantiates object models in the world model. Figure 2-3 shows the appearance of sensors on the platform. There are multiple cameras; a directable range finder which is bore-sighted with a camera in a central location; a tactile probe; and a fixed sensor pod for measuring ambient properties of the environment. The ideal sensor systems on the telerobot would support:

- multiple CCD cameras which can be moved independently with high precision.
- the ability to determine three dimensional positional information in a precisely specified direction
- the local orientation of the robot platform relative to the immediate ground plane and gravity
- the ability to determine the moisture characteristics of the atmosphere, the temperature, the position of the sun

- tactile probes to establish the characteristics of the immediately surrounding terrain and soil.

A directable range finder would be useful though multicamera stereo will also be able to supply depth information. We feel that one which can get precise depth information at selected environmental points will be of equal or more value than a scanning laser range finder. Figure 2-4 shows a proposed design for attaching a laser range finder to a CCD camera such that the range to selected points could be determined.

While the reliance on active range sensors for the telerobotic platform would simplify perceptual processing, it will be of limited value in military operations where not being detected is crucial. In addition, current active sensors have some limitation when operating in complex, natural environments. For this reason, passive sensing techniques, such as multicamera stereo, will need to be utilized.

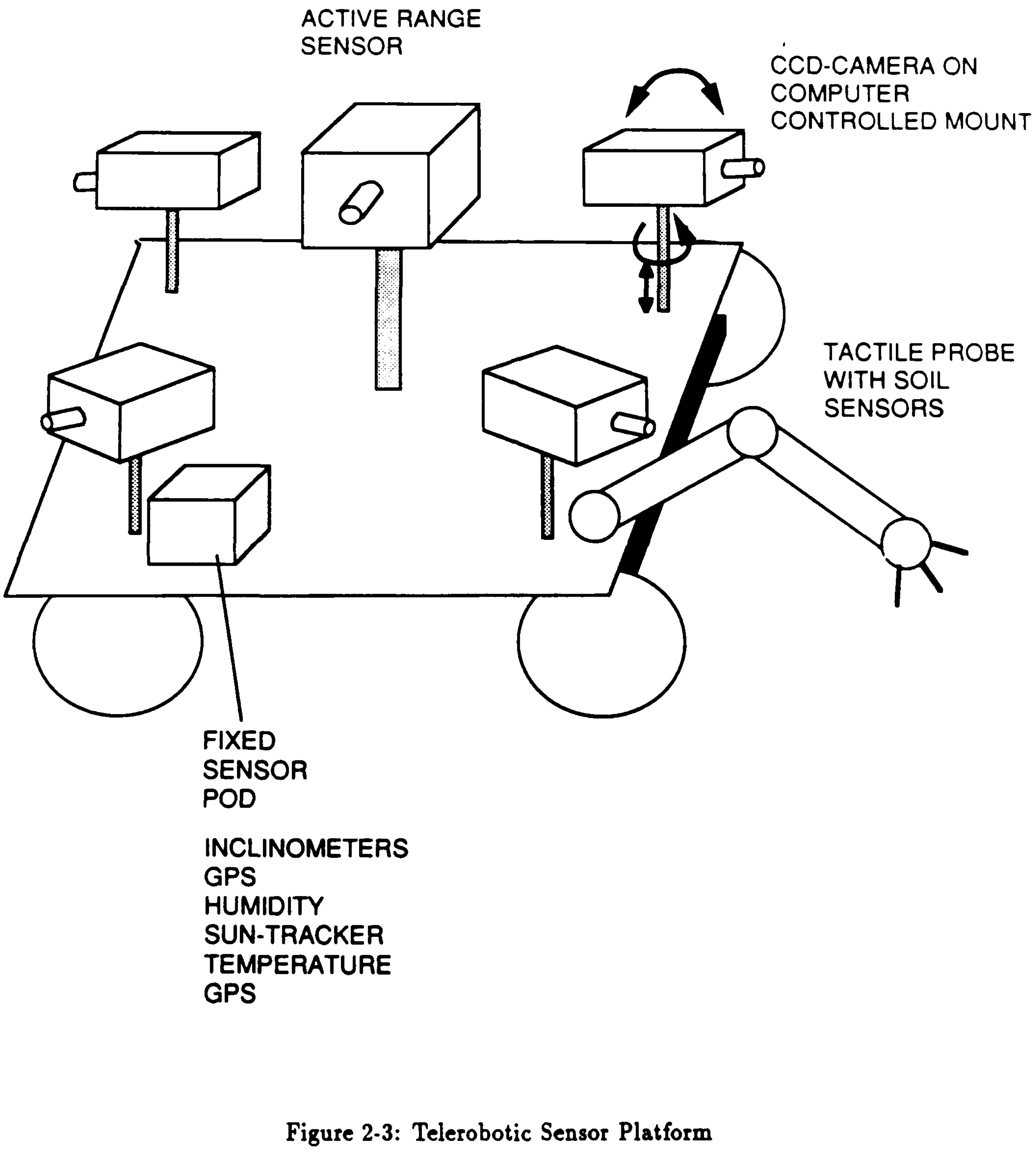


Figure 2-3: Telerobotic Sensor Platform

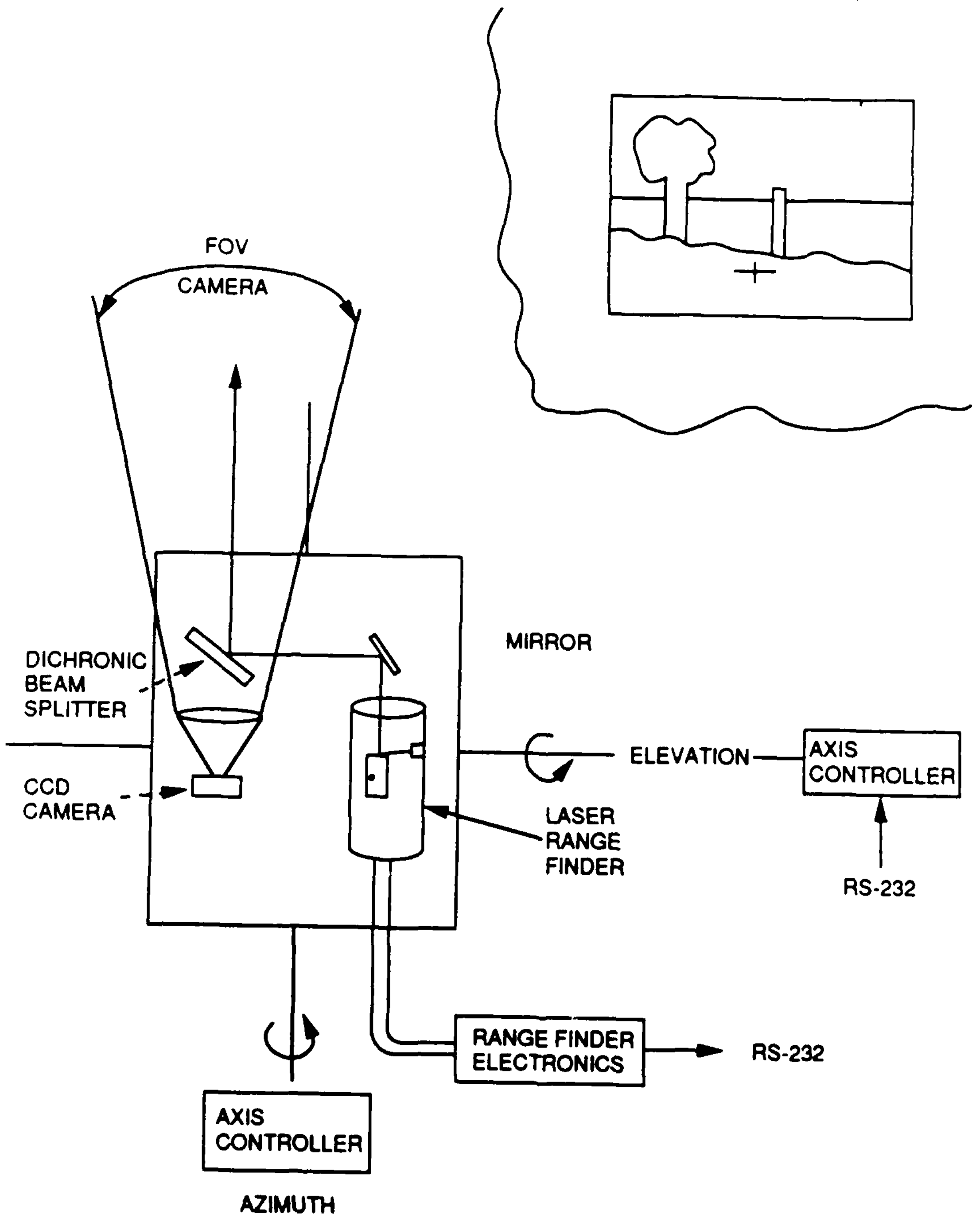


Figure 2-4: Camera with Boresighted Laser

3. Constraint-Based Representations and Reasoning

Constraint-based representations allow us to define an object as a set of assertions and relationships that are required to be true for the object. These assertions will generally reflect real world relationships. Constraint-based representations can be manipulated with considerable flexibility. This is especially true for our applications because they do not have any preset directionality in the order in which attributes for an object are determined when it is instantiated. This allows a human to change a small number of attributes of an object interactively without having to change all the effected attributes.

A simple example of this is shown in Figure 3-1 where an object is defined to be a square in terms of its required relationships for the lengths of its sides all being equal and all its angles being 90 degrees. When a human interactively grabs a point on the square and moves it, we'd like the rest of the points to change is such a way that the figure remains a square and doesn't turn into some type of quadrilateral. In this way, the human can specify a small amount of information and have automatic maintenance of consistency. This type of control is easily expressed using constraint based techniques. It is very difficult to express this in a typical graphics packages where objects are defined in terms of the positions of points and not their required relationships. In the bottom of Figure 3-1 we see some of the corners darkened. This corresponds to setting the position of the corners so they can not be moved. In this case, when the human would interactively try to move the unrestricted corner, a constraint-based system would respond with the impossibility of doing this without violating the definition of a square.

In this section, we describe the general properties and current limitations of constraint-based representations. We then describe how they should be utilized to direct the pick-up of information and the form of object models.

3.1 Properties of Constraint-Based Reasoning and Representation

As described above, a constraint is a statement about what must be true. In general, a constraint can apply to the values of one or more variables. If a constraint applies to a single variable, then the constraint expresses something that must be true of the value that will be assigned to that variable. For example, take the constraint `red(X)` (lower case names are constants, upper case names are variables). This constraint says that, whatever thing `X` is eventually made to point to, that thing must be red. If a constraint applies to two or more variables, then the constraint expresses a relationship that must hold between the values of the constrained variables. As another example, the constraint `brother(X,Y)` requires that, whatever things `X` and `Y` are made to point to, those things must be brothers. As a third example of a constraint we might have $V = L^3$ which relates the volume of a cube to the length

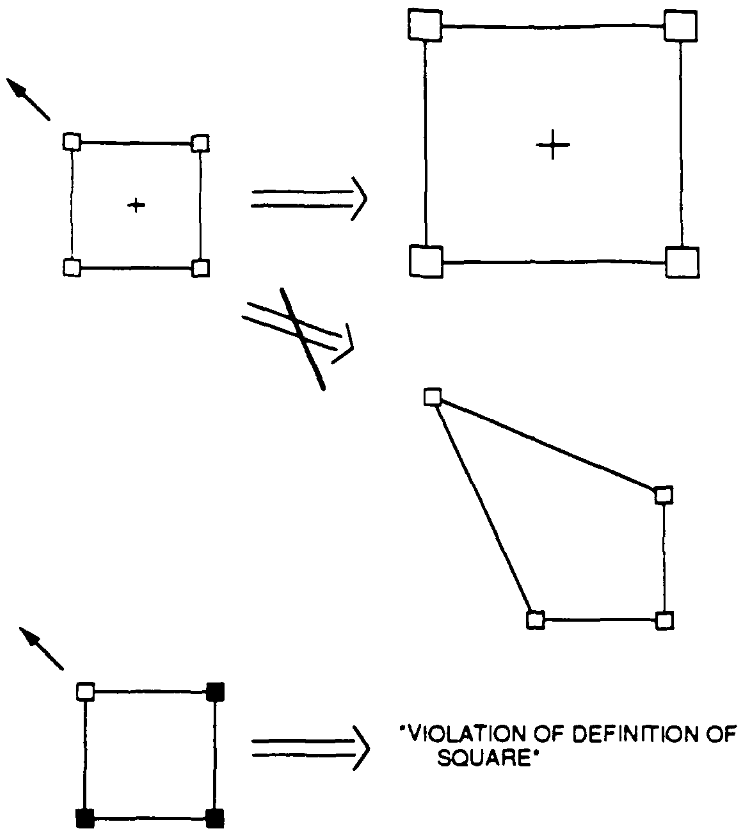


Figure 3-1: Importance of Constraints

of its edges. Finally, Ohm's Law $V = I \times R$ is a constraint that relates the values of the voltage, current and resistance across a given conductor.

Constraints express knowledge about the world, whether or not the variables mentioned by the constraint have values assigned to them. The law about the volume of a cube is a clear example of knowledge of a physical principle that can be applied to any and every cube, no matter what its size. Similarly, the constraint `brother(X,Y)` says something about how `X` and `Y` must be related whether or not we know exactly what (or who) `X` and `Y` might be.

This is very a useful capability for any problem solving program. Suppose we had a program that was looking for specific people to fill two positions in a company. The `brother` constraint would allow our program to say what it knew about those people, without having to prematurely *name* two specific people that happened to be brothers. In fact, if the program merely picked out two people that were brothers and gave us their names we might not know that they *were* brothers. That is, the constraint conveys information even when it applies to specific objects.

Furthermore, the knowledge that the two people required for the job had to be brothers would be important information in making the selection: it would rule out the vast majority of applicants and allow for a much more efficient search. Instead of considering the company's every employee for both of the new positions, the program could consider only those employees who had brothers that also worked for the same company. Evaluating only those few people would make the program far more efficient. The usefulness of constraints for efficient searching was demonstrated most forcefully by the MOLGEN system [Stefik - 81].

In fact, most of human knowledge is exactly in the form of constraints. When I describe a cube I say that its twelve edges are all the same length. That is a constraint, because it can specify a relationship between the lengths of the twelve edges without saying how long the edges actually are. Also in a cube, the angle between every pair of touching edges should be a right angle. That is also a constraint because it specifies the size of angles that could have any other size. Again, in a cube the volume is the third power of the edge length, whether or not the edge length is known.

3.2 Adirectional Constraint Satisfaction

Constraints are not only statements about what must be true; they also allow the deduction of unknown values from known ones. Take the case of the cube. If I tell you that a particular edge is 6" long you can immediately deduce that the other eleven edges are also 6" long. Further, you can deduce that the volume of the cube is 6"×6"×6".

By telling you the edge length I gave you enough information to allow you to *satisfy* the constraint between the volume and edge length of a cube. A constraint is satisfied when it has values (or bindings, or assignments) for every variable mentioned by the constraint. In the case of constraints that happen to be equations (such as $V = L^3$) you need values for all but one of the variables, and given those values you can then compute the value for the outstanding variable. You can do something similar for the `brother(X,Y)` constraint, however. If I tell you the name of one brother you can probably tell me the name of the other, especially if there are only two brothers in that family.

An important and useful point is that constraints are often *a-directional*. This means that, even when there is only one set of values that will satisfy the constraint, that set of values can be computed in several ways. To take the cube example again, above you computed the volume of the cube when given the cube's edge length. On the other hand, I could decide to tell you the cube's volume, in which case you could compute its edge length. Either way you would have satisfied the constraint relating volume to edge length, but the important point is that a single constraint can generally be used in many ways. The Ohm's Law constraint is even more flexible. $V = I \times R$ can be satisfied in three ways:

- If you know I and R you can compute V.

- If you know V and R you can compute I.
- If you know V and I you can compute R.

Constraints of this kind were used by the EL system [Stallman and Sussman - 78] to design electrical circuits.

The a-directionality feature of constraints points to another useful aspect of constraint-based computing, namely that a check on a constraint's readiness for satisfaction can be made part of the constraint itself. That is, whenever the constraint is supplied with enough information to allow deduction of the outstanding value, the constraint can notice that fact and perform the deduction without having to be asked. This means that, no matter what information becomes available to the constraint-based program, and no matter in what order that information arrives, the program will make the possible deductions without having to be asked to do so. In other words, a constraint-based program can be made to behave like a dataflow machine, or like a data-driven forward-chained inference engine. This is extremely convenient when the information being provided to the program is coming from an interactive terminal, i.e. its order is entirely unpredictable. Since the programmer who is encoding the constraints does not have to consider the order in which information will arrive, nor when to make any deductions, nor when to ask for the results, the programming task is made much easier.

3.2.1 History of Constraints

One of the founding contributions to work on constraint satisfaction was SKETCHPAD [Sutherland - 63], a general-purpose system for drawing and editing pictures on a computer. In SKETCHPAD the user interacts directly with the display, using a light pen for adding, moving, and deleting parts of the drawing.

ThingLab is the system closest to our own intentions. ThingLab [Borning - 79] adopted much of SKETCHPAD's user interaction, as well as its notions of constraints and of recursive merging. Of course, ThingLab also extended SKETCHPAD's constraint mechanism, by integrating it with an inheritance hierarchy, and by allowing local procedures for satisfying a constraint to be included as part of its definition. Many of the important ideas in Smalltalk also appear in ThingLab – objects, classes and instances, and messages – and will be used in our own work.

We will also exploit the work on constraint languages done at MIT by Sussman and Steele [Steele, Jr. - 80, Steele, Jr. and Sussman - 78]. Our implementation of constraints will be very similar to theirs, including their dependency maintenance capabilities. Thus we will provide graphics and inheritance as well as dependency directed backtracking.

Other related languages include SIMULA [Dahl and Nygaard - 66], MESA [Mitchell, Maybury, and Sweet - 79], CLU [Liskov et al. - 77], and ALPHARD [Wulf, London, and Shaw - 76]. These languages separate the interface specification of a type from its internal implementation, just as Smalltalk distinguishes the external message protocol of an object form its internal aspects. Thus, in programs in these data-abstraction languages, changes to the implementation of a type (but not its interface) do not affect the users of that type; so more modular systems result.

Work in Artificial Intelligence includes a number of systems that use constraints and constraint satisfaction as such. Steels [Steels - 79] has constructed a reasoning system, modeled on society of communicating experts, that uses propagation of constraints in its reasoning process. Unlike either ThingLab or Sussman's work, Steels' system is description-oriented and does not require that constraint satisfaction yield a unique value. Stefik [Stefik - 81] uses constraint satisfaction in a hierarchical planner for molecular genetics experiments.

3.2.2 Constraints in Graphics Systems

There have been a small number of systems that combined the use of constraints with display graphics, namely SKETCHPAD [Sutherland - 63], ThingLab [Borning - 81] [Borning - 79], IDEAL (previously called LELAND) [Van Wyk - 80] and Bertrand [Leler - 88]. However, in all of these systems the data structures and constraints were designed for two-dimensional space, and none incorporated dependency maintenance and retraction. The focus of these systems was on the creation of planar diagrams and circuits by specifying them with appropriate constraints. Our emphasis is on finding sets of constraints that explain perspective images of prior three-dimensional scenes. The extra dimension makes the constraint satisfaction problem considerably larger and more difficult, and our emphasis on interpretation as constraint *selection* is altogether new.

3.3 An Example

It is time to show an example of constraint based reasoning. First we specify the "domain constraints" that describe a world containing 3 blocks of equal size. As in PROLOG, atoms beginning with capital letters are variables, atoms beginning with small letters are constants or predicate names. Merely for the sake of notational convenience I have written the constraints with the assumption that different variables must be bound to different objects.

```
;; circumscribe what can be on a block.
on(a,X) ∨ on(b,X) ∨ on(c,X) ∨ clear(X)
```

```
;; blocks can have only one thing at a time on them.
¬on(X,A) ∨ ¬on(Y,A)
¬on(X,A) ∨ ¬clear(A)

;; circumscribe what a block can be on.
on(X,a) ∨ on(X,b) ∨ on(X,c) ∨ ontable(X)

;; blocks can be on only one thing at a time.
¬on(A,X) ∨ ¬on(A,Y)
¬on(A,X) ∨ ¬ontable(A)

;; rule out circular towers of one block.
¬on(A,A)

;; rule out circular towers of two blocks.
¬on(A,B) ∨ ¬on(B,A)

;; there's always something on the table.
ontable(a) ∨ ontable(b) ∨ ontable(c)
```

This defines a network of constraints. Whenever new information is added to this constraint set, the reasoning system uses the new information to satisfy as many constraints as possible. The example below shows how each new piece of information allows the reasoning system to infer more about the arrangement of the blocks being described.

at this point the reasoning system knows only the general rules, so knows nothing about a particular arrangement of the three blocks.

```
> on(a,b)f        input from the user

=> ¬clear(b)      what isn't on b
=> ¬on(c,b)
=> ¬on(a,c)       what a isn't on
=> ¬ontable(a)
=> ¬on(b,a)       no circular 2-towers
```

at this point, block c *might be under block* b *, or it might be on top of block* a *, or it might be off by itself.*

```
> clear(c)        input from the user

=> ¬on(b,c)       new conclusions...
=> ontable(b)     because b isn't on a or c
```

at this point, block c *might be on top of block* a, *or it might be off by itself.*

```
> ¬ontable(c)   input from the user

=> on(c,a)      new conclusions...
=> ¬clear(a)
```

This example is useful partly for showing how constraint satisfaction works, and partly because the constraints being satisfied are logical rules that constrain the *truth values* of various facts. For example, when the user first asserts that on(a,b) is true, the reasoning system notices that ¬on(a,b) is false, that ¬on(a,b) is a special case of ¬on(X,A), and that to keep the constraint

$$\neg\texttt{on(X,A)} \lor \neg\texttt{clear(A)}$$

true, ¬clear(A) had better be true. Since this constraint was matched with A = b the system deduces that ¬clear(b). All the other inferences are made in exactly the same way. But notice that what is being constrained here is not the variables X and A, but the *truth values of facts*, i.e. whether facts such as clear(b) are true or false. The variables being constrained are just the boolean variables that represent the truth values of those facts. This is a special case of constraint satisfaction, and this special case has been studied under the heading "Truth Maintenance Systems" [Doyle - 79, McAllester - 78].

3.4 Dependency Maintenance and Dataflow

What if the information given to the reasoning system was wrong? Or what if we were guessing, and now we want to try a different possibility? Using a technique called "dependency maintenance", a constraint reasoning system can keep track of how various conclusions depend on each other. Hence when we withdraw an assertion on which other conclusions were based, those derived conclusions can also be withdrawn.

Let us continue the block stacking example used above. We told the reasoning system that on(a,b) and clear(c) and ¬ontable(c). From this it deduced that block b had to be on the table, and that c had to be located on top of block a. Suppose we decide to withdraw the assertion that on(a,b). Here is how the reasoning system responds:

at this point the reasoning system believes that on(a,b) *and* clear(c) *and* ¬ontable(c).

```
> retract on(a,b)   input from the user

-- ¬clear(b)
-- ¬on(c,b)
-- ¬ontable(a)
-- ¬on(b,a)
-- ontable(b)
-- on(c,a)
-- ¬clear(a)
```

The reasoning system assumes that clear(c) and ¬ontable(c) are to remain true because we haven't retracted them. As a result, conclusions that follow from those facts *alone* are still believed (for example: that ¬on(a,c) and ¬on(b,c)). However, all of the conclusions that depended on the truth of on(a,b) have been withdrawn.

Notice that the retraction could be done even though the fact we retracted was not the last one to be asserted. This was possible because the reasoning system kept track of the dependencies between conclusions, and because it is the dependency structure (rather than the order of our assertions) that determines what must be retracted.

Now that block a is no longer on top of block b, we are free to put blocks a and b somewhere else. Let us put block b on top of block a instead:

at this point the reasoning system believes that clear(c) *and* ¬ontable(c).

```
> on(b,a)         input from the user

=> ¬clear(a)
=> ¬on(c,a)
=> ¬ontable(b)
=> ¬on(a,b)
=> ontable(a)
=> ¬clear(b)
=> on(c,b)
```

The system has now changed its mind about many of the conclusions that held when on(a,b): what was true then is largely false now, and vice versa.

Similar reasoning applies in the case of the constraint relating a cube's volume to its edge length. When we tell the constraint reasoning system that the cube's edge is a certain length, it computes the cube's volume based on that information. If we change withdraw our statement about edge length the system retracts the conclusion about volume. If we then assert a new edge length the system will conclude a different volume.

Consequently it is not hard to make a constraint reasoning system behave like a spreadsheet, or like a dataflow computer. Whatever the system is given as input determines its conclusions. If we change one of the inputs the system revises all the outputs that depend on that input. At all times the output values reflect the input values. Nevertheless, a constraint reasoning system is somewhat more general than a spreadsheet because constraints are a-directional (as discussed above). In other words, we can tell a constraint reasoning system what a column should add up to, and unlike a spreadsheet, the constraint reasoning system can fill in a missing entry in the column.

Figure 3-2 shows how the spreadsheet-like behavior of constraint systems can be put to good use in graphics applications. In this Figure there are eight lines and four points. Four of the lines are constrained to meet end-to-end, and so form the outer quadrilateral. The four special points are constrained to bisect the lines of the outer quadrilateral. The four remaining lines are constrained to form another quadrilateral that has the four special points as its corners. With all those constraints in place, the user can manipulate the shape of the outer quadrilateral while the constraint reasoning system sees to it that all the constraints remain satisfied. That is, the new positions for the midpoints and for the smaller inner quadrilateral are automatically recomputed using the dependencies set up by the constraints. The constraint reasoning system thus allows the user to *see* that, no matter what he does to the shape of the outer quadrilateral, the inner quadrilateral is always a rectangle.

3.5 Higher Order Constraints

The constraints I have described so far have been rather simple. They have consisted either of arithmetic linear equations, or of logical implications. It turns out that there are practical problems these constraints cannot handle. In particular, there are situations in which the constraints could be satisfied because all the necessary information is present, but the constraint reasoning system cannot detect that fact. This weakness in a reasoning system is called "incompleteness".

Incompleteness occurred almost as soon as constraint satisfaction was invented. The EL system [Stallman and Sussman - 78], in its reasoning about circuits, came across resistive voltage dividers as shown in Figure 3-3. An Ohm's Law constraint applied to each of the two resistors R1 and R2. Even though the reasoning system knew the values of the two resistors and knew that the current flowing through the two resistors was the same, it could not work out the value of that current. The problem was that the system did not know how to relate the values of the individual resistors to the total resistance of the circuit. That is, the system did not have the

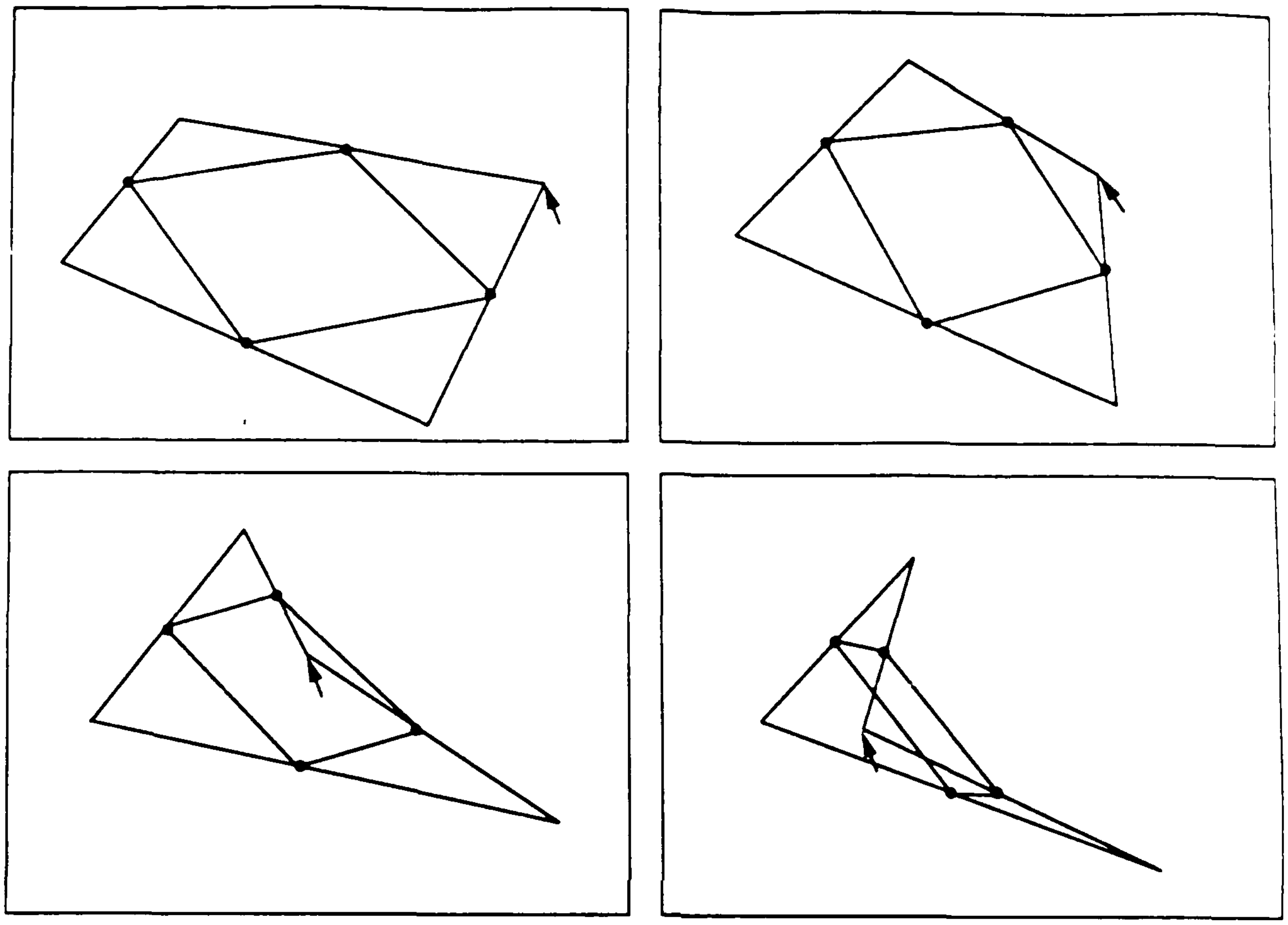


Figure 3-2: Constrained Graphics Manipulation

concept of resistors-in-series. The solution was to define a new kind of constraint that specialized Ohm's Law to resistive voltage dividers. This new constraint was $V = I \times (R1 + R2)$. Given this constraint, EL could use the values of V, $R1$ and $R2$ to compute I, and this value of I was then used by the originally existing constraints to compute the voltages across the individual resistors. The new constraint was described as containing a "redundant view" of the circuit.

It is our view that the new constraint was not "redundant". Indeed, it embodied the new information that the total resistance of two resistors in series is just the sum of their resistances. This is important and useful information. However, EL could not use this information in a general way. Resistors may occur in series whether or not they are forming a voltage divider. Indeed, resistors may occur in series whether or not they were explicitly required to be in series. But EL could reason about series resistors only if their seriality was intentional; EL could not *notice* that two resistors *happened* to be in series.

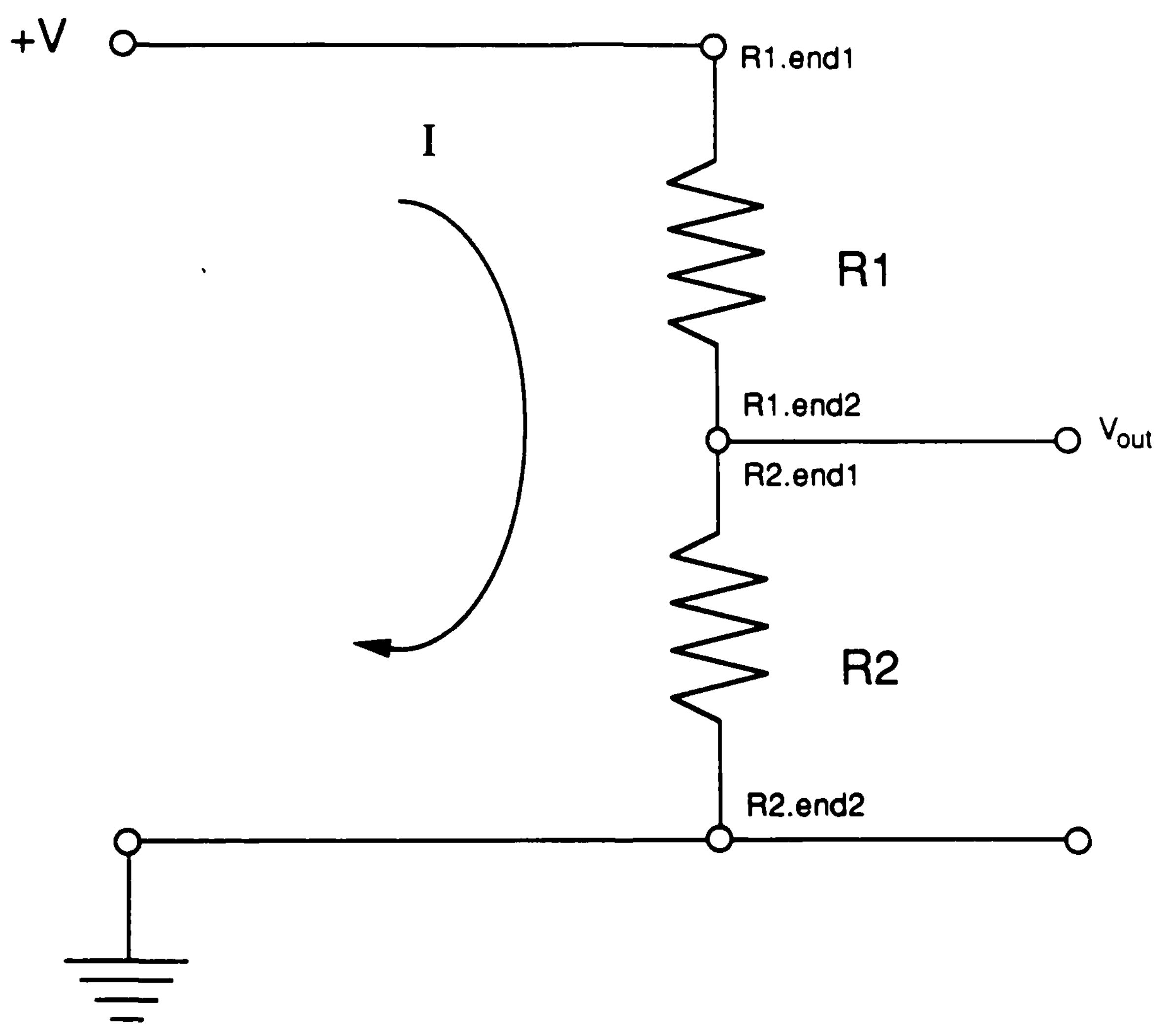


Figure 3-3: A Resistive Voltage Divider

Related to the incompleteness problem in constraint reasoning is the issue of reasoning over both logical constraints and arithmetic constraints. For example, it becomes possible to write such constraints as

```
resistor(R1) ∧ resistor(R2) ∧
  (R1.end2 = R2.end1) ∧ (R1.current = R2.current)            (C1)

    ⊃ R1.current = (R1.end1.voltage - R2.end2.voltage)
                       / (R1.resistance  R2.resistance)-
```

This is a logical constraint that relates the truth values of several arithmetic constraints. More generally, constraints that constrain the truth values of other constraints are called "higher order" constraints. At present BERTRAND [Leler - 88] is the only constraint satisfaction system that allows higher order constraints of this form.

The higher order constraint just shown (C1) would would solve the voltage divider problem. The constraint says that *if* two resistors happen coincidentally to be connected end-to-end, and if no current flows out of the terminal connecting them, then the current (through both of them) may be computed by using Ohm's Law as if the two resistors were one.

Note that even constraint (C1) does not do justice to the idea of resistors-in-series: it allows the constraint reasoning system to view the resistors as being in series *for the sole purpose* of applying Ohm's Law to the series. If the voltage divider happened to consist of three resistors in series, the reasoning system would be entitled to view only two of the three as being in series, and hence would still not be able to solve for the current flowing through the series. To properly capture the idea that two resistors in series may be treated as a single resistor, we would need constraints such as:

```
resistor(R1) ∧ resistor(R2) ∧
  (R1.end2 = R2.end1) ∧ (R1.current = R2.current)
      ⊃ ∃RX.series-resistor(R1,R2,RX)
series-resistor(R1,R2,Rs) ⊃
      resistor(Rs) ∧                                 (C2)
      Rs.end1 = R1.end1 ∧
      Rs.end2 = R2.end2 ∧
      Rs.current = R1.current ∧
      Rs.current = R2.current ∧
      Rs.voltage = R1.voltage + R2.voltage
```

That is, these constraints would create an entirely new conceptual object RX in the circuit, effectively defining a new resistor that had all the properties of a normal individual resistor (including the use of Ohm's Law). This approach could handle an arbitrary number of resistors in series: as a pair of resistors R1 and R2 was subsumed by a series-resistor RX1, the new series-resistor could be regarded as an individual resistor in series with the next adjacent resistor R3, and could be subsumed in turn.

The ability to have constraints create new objects on an opportunistic basis does not exist in any extant constraint reasoning system. There is however an alternative approach that both recognizes the contingent nature of higher order constraints, and that allows for a more general approach to constraint satisfaction.

3.6 Selection Procedures

It is possible to separate

- recognition that a constraint can be satisfied, from
- calculation of the satisfying value for the last unbound variable.

Indeed, all constraint systems do this to some extent (see especially THINGLAB [Borning - 79]). For the Ohm's Law constraint $V = I \times R$, for example, the value of V must be computed using a multiplication operation, while the value of R must be computed using a division operation. This happens to be a simple example: the central operation is easily invertible. Similarly, for the cube's volume equation $V = L^3$, V is computed from L by cubing, L is computed from V by taking the cube root. For such invertible operations, constraint systems include both the operation and its inverse under a single notation, but in general both the operation and its inverse must be present in the *realization* of the constraint. That is, depending on which variable is being bound to satisfy the constraint, the reasoning system may run a different operation with the constraint's remaining variables as inputs. The operation being used to bind the last variable is a "satisfaction procedure".

While it is a convenience that for arithmetic constraints many satisfaction procedures can be inferred from the operations used to define the constraint, this convenience does not even suffice for very much of mathematics. The constraint $Y = a \times X^2 + b \times X + c$ allows for computing the value of Y from that of X, and can be inverted to compute the value of X from that of Y - the inversion is the well-known "quadratic equation" - but no existing constraint satisfaction system is capable of discovering that inversion for itself. To allow the system to compute X from a known Y,[1] the system's user must inform the system of the quadratic equation as a redundant constraint. In effect the user is supplying the constraint system with a satisfaction procedure the system could not figure out for itself.

[1]Some constraint satisfaction systems provide a technique called "relaxation" in which solutions are guessed at and refined with iteration, as in Newton's method for finding roots. Relaxation is not guaranteed to find solutions to nonlinear or trigonometric equations.

But satisfaction procedures are not yet general enough to solve the voltage-divider problem discussed in the previous section. The problem with the voltage divider was not that the reasoning system did not know how to invert an individual constraint. Rather, the problem was that the system lacked certain critical knowledge, i.e. that it was possible to view two resistors in series as a single resistance for the purpose of computing the voltage drop across the individual resistors. In other words, *despite* the system's knowing how to invert all the initially given constraints it lacked the knowledge that a particular *configuration* of constraints could be useful.

Satisfaction procedures recognize that the operation used to bind a constrained variable generally depends on both the particular constraint and the variable being bound. Nevertheless, satisfaction procedures are still derived from individual constraints. Consequently they fail to recognize that a conjunction of constraints on the same variable might allow a solution method that could not be used unless *all* those constraints wer .esent.

"Selection procedures" take the idea of satisfaction procedures one step further: selection procedures are organized around variables rather than around constraints, and the particular method employed by a selection procedure when it is binding a variable becomes explicitly contingent on what constraints are present.

Selection procedures can solve the voltage-divider problem by recognizing the existence of a resistive voltage divider, and by knowing how to compute the output voltage in such a circumstance. Specifically, computing the output voltage requires knowing a reference voltage (such as the battery or a ground) and a potential difference across one of the two resistors. Knowing the potential difference requires knowing the current through the chosen resistor. Knowing that current requires knowing the total resistance of, and voltage across, the voltage divider configuration. Knowing the total resistance requires addition of the resistors in series. Thus the selection procedure for the output voltage behaves differently depending on whether or not a voltage divider configuration exists.

Selection procedures are extremely useful in image interpretation because the information required to compute some result may be distributed across many objects in an image. Consider the example shown in Figure 3-4, where an image interpretation system knows the position and orientation of one camera relative to the other, and also knows the size of an object visible to both cameras. Given the angles β_i subtended by the object at the two cameras, as well as the angles α_i between the object and the camera axes, the system could – in principle – compute the position and orientation of the object (this is simply a triangulation problem). Yet it is a nontrivial matter to have a constraint satisfaction system realize that such triangulation is possible. The difficulty is that the necessary parameters must come from different places in the scene model – specifically, the parameters must be provided by two cameras and the observed object. Using the standard approach to constraints it would be necessary to create one "triangulation constraint" for each possible combination of camera-

1, camera-2 and observed-object! Of course, the space required to make all those constraints explicit would be impractically large.

To solve the same problem with selection procedures, the observation that triangulation was possible would be made by a selection procedure capable of computing an object's position and orientation in space.

In summary, selection procedures reorganize the knowledge required to satisfy constraints. Where previous constraint satisfaction systems have organized constraint satisfaction knowledge around the constraints themselves, our image interpretation application has led us organize constraint satisfaction knowledge around the variables being bound. In this way, the coincidental presence of two or more constraints on the same variable may allow for solution methods that could not be noticed by any of the constraints taken alone.

3.7 Solving Systems of Equations

The most general view of constraint satisfaction - when all the constraints are arithmetic - is as solving a system of equations. Equation solving can be made arbitrarily difficult if the equations can be nonlinear, differential or integral, trigonometric or transcendental. General equation solvers such as MACSYMA and MATHEMATICA are very difficult to build. They are in fact expert systems with mathematics as their domain of expertise. For many applications they are too powerful as well as being too slow. Thus constraint satisfaction systems might be seen as a special class of equation solving system that is exceptionally efficient by virtue of the limitations placed on the kinds of problems that can be solved. Selection procedures serve to pull constraint satisfaction systems a little further in the direction of generality without a significant loss of speed. In particular, selection procedures may be built to solve frequently occurring conjunctions of equations/constraints in such cases as the resistive voltage divider and triangulation. This is still a far cry from general equation solving but will probably be adequate for our purposes.

Conversely, constraint satisfaction systems can provide functionality that equation solvers can not provide. The most obvious such functionality is that of dependency maintenance. Constraint satisfaction systems, particularly those that do not rewrite the equations they are solving, can reuse those equations when their input values are changed. This is crucial for systems that may wish to experiment with appropriate values for variables, and that may have to retract the results of previous experiments. Image interpretation is an application in which such guessing and retraction may be very frequent, so that constraint satisfaction systems are preferred over equation solvers.

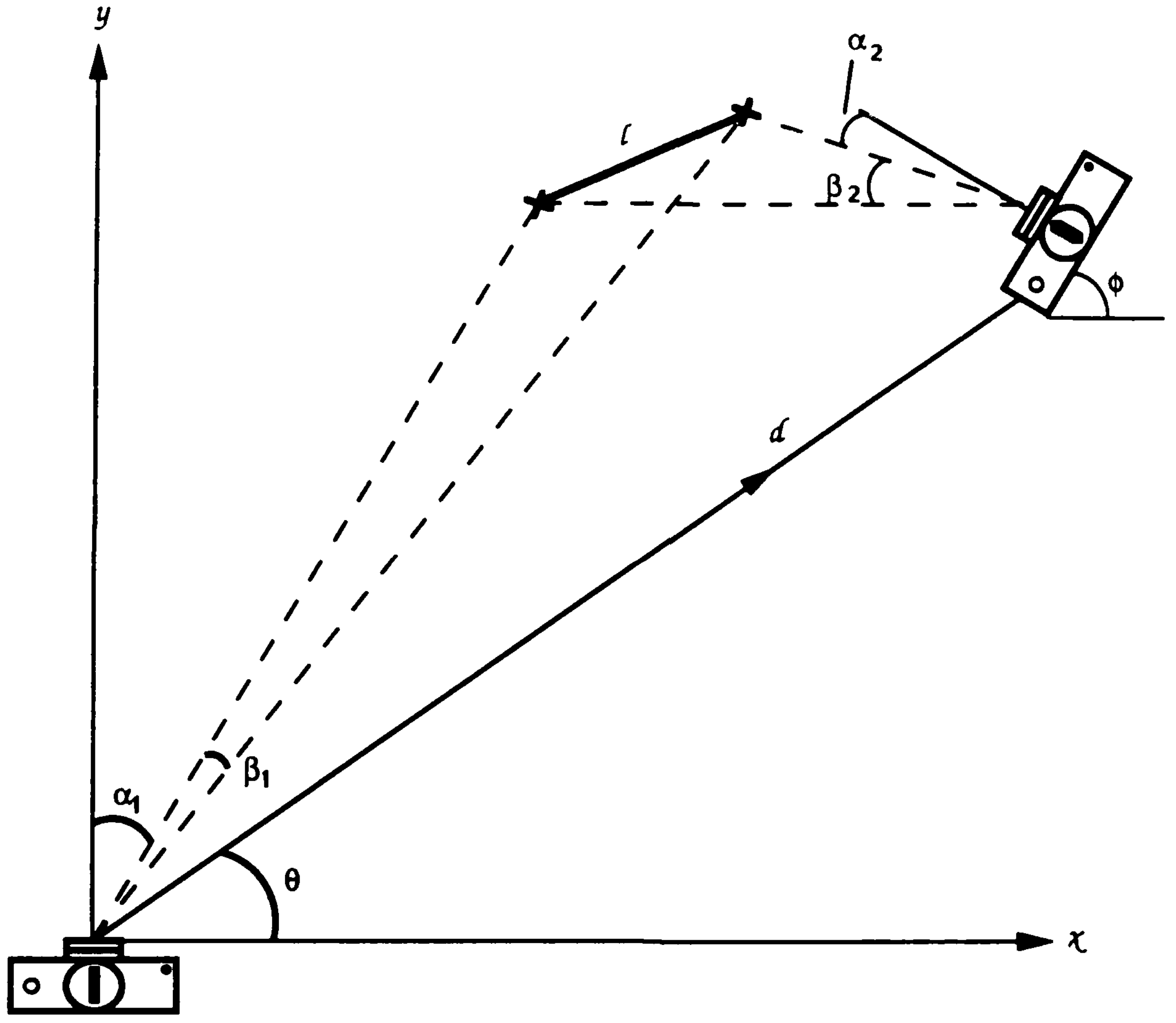


Figure 3-4: Triangulation as a Selection Procedure.

3.8 Implementing Constraint Choosing

We suggest that image interpretation be regarded as a process of choosing a set of constraints that

1. account for what is visible or occluded in the image, and
2. allow for a consistent assignment of values to variables.

Accounting for what is visible or occluded in the image is a process of hypothesis formation: the set of constraints chosen to explain the image is a theory that reconstructs the scene that the image represents. The existence of a consistent assignment (or satisfaction) is a way of checking that the theory is physically plausible, that the hypothesized scene could be a real one.

In this Section we explain how constraints are chosen and satisfied. We begin by sketching the data structures required. In particular, the objects visible in the image become computational "objects" in the image interpreter. We then describe how the constraint management system can assist the human user by pointing out interpretation errors.

3.8.1 Image Objects as "Objects"

Computational "objects" are records with access procedures attached. Said differently, "objects" are groups of variables, but are constructed in such a way that it is not possible to read or write the values of those variables directly. Instead, if our program wants to change the value of some variable inside an object, then our program must ask the *object* to execute a procedure that modifies the specified variable to its new value. While this might seem a round-about way of doing things it has some benefits in that the object may decide not only to update the named variable as requested, but may perform numerous other functions as well. Perhaps the object needs to ensure some kind of consistency of the new value with the values of other variables in the object. In this way the object can prevent the user's program from making certain kinds of errors. Equivalently, object-oriented programming can lift some of the consistency burden off the user.

To show the useful features of objects we describe the data declarations for a program that reasons about planar quadrilaterals (closed four-sided figures).

We begin by defining vectors as triples of real numbers (the base of the vector is taken to be the origin of the coordinate system). Then line segments are constructed as pairs of vectors. The notation we use here is not part of any programming language; we have chosen it to make our intent clear.

```
define vector

    parts

        x, y, z:  real;
        l, a, e:  real;

    constraints

        l² =x²+y²+z²;

        x = l cos(e) cos(a);
        y = l cos(e) sin(a);
        z = l sin(e);
        y = x tan(a);
        z = x sec(a) tan(e);

end vector;

define segment

    parts

        p1, p2:  vector;
        v:  vector;

    constraints

        p2.x = p1.x + v.x;
        p2.y = p1.y + v.y;
        p2.z = p1.z + v.z;
        ...

end segment;

define angle

    parts

        v1, v2:  vector;
        a:  real;

    constraints

        cos(a) = sin(v1.e) sin(v2.e) +
                 cos(v1.e) cos(v2.e) cos(v1.a - v2.a);

end angle;
```

This definition of a vector recognizes that there are two useful ways of specifying vectors. The first is as a displacement along three Cartesian axes; hence the x, y, z parts of the vector. The second way is in Polar coordinates, giving the vector a length, an azimuth and an elevation; hence the l, a, e parts of the vector. The vector must also maintain constraints between the two sets of coordinates. For example, if the z component of a vector is changed, the vector's length or elevation may also change and must be updated if they are to remain consistent with the new z value. This is one example of the utility of the access restrictions that come with object-oriented programming: the object's z value can only be changed by requesting the object to do so, at which point the object cannot be prevented from updating several other variables as well.

The above definition of a line segment recognizes that there are two ways to specify any line segment. One way is to specify the two endpoints of the line. The other way is to specify one endpoint and a relative displacement, so that the other endpoint can be computed. Of course, given the two endpoints we might become interested in the line segment's length, i.e. in the length of the vector between the endpoints. The constraints of the line segment ensure that no matter which way the line segment is originally specified, the other way will be accessible, and the two will be kept consistent with each other.

The above definition of an angle computes the angle between the two vectors referred to by the object. This object exists primarily for the constraint it carries.

We are now in a position to define a general quadrilateral. We begin by defining an arbitrary four-sided figure (assumed to be planar; we will not go into serious detail here). The four sides of a quadrilateral must join end-to-end to form a closed figure, and there are four angles between the adjacent sides.

```
define quad
    parts
        s1, s2, s3, s4:  segment;
        a12, a23, a34, a41:  angle;
    constraints
        s1.p2 = s2.p1;
        s2.p2 = s3.p1;
        s3.p2 = s4.p1;
        s4.p2 = s1.p1;
        a12.v1 = s1.v; a12.v2 = s2.v;
        a23.v1 = s2.v; a23.v2 = s3.v;
        a34.v1 = s3.v; a34.v2 = s4.v;
        a41.v1 = s4.v; a41.v2 = s1.v;
end quad;
```

Next, by constraining that two opposite sides be parallel, we can specialize to get a trapezium. We define a trapezium as a special case of a quadrilateral, as follows:

```
define trapezium from quad

     constraints
          s1.v.a = s3.v.a;
          s1.v.e = s3.v.e;

end trapezium;
```

Because the trapezium is defined **from** a quadrilateral, it has all the same properties as quadrilaterals. In particular, trapeziums will have four sides and four angles, and those parts will have the same names as in a quadrilateral, so they need not be defined again. In object-oriented language, the trapezium class "inherits" from the quadrilateral class.

To complete the trapezium definition we constrain sides s1 and s3 to be parallel. To be precise, we access side s1's line segment vector v (as opposed to s1's endpoint vectors p1 and p2), and then the azimuth of that line segment vector. Similarly we access side s3's line segment vector's azimuth, and make the two equal. Likewise for the elevation angles.

To continue: we can add successive constraints to various types of four-sided figures to define, in turn, parallelograms, rectangles, rhombuses and squares. (Figure 3-5). Each type of four-sided figure inherits all the parts and constraints of the types above it. Merely the addition of one or two new constraints specializes each type of quadrilateral into the type(s) below it. Underneath parallelograms we can choose to constrain all angles to be right angles, which gives us rectangles; or we can choose to constrain all sides to have equal length, which gives us rhombuses. If we add both sets of constraints at once we get squares. That is, squares are special cases of both rectangles and rhombuses, so inherit constraints from both. This is called "multiple inheritance".

The chief value of inheritance is that it allows programmers to define objects incrementally, i.e. by making small modifications to previous definitions. It is very easy to define a rectangle as a right-angled parallelogram, for example, or a square as something that is both a rectangle and a rhombus.

3.8.2 How to Satisfy Constraints

So far we have seen that constraints are actually collections of selection procedures: the constraint $V = I \times R$ is a collection of three selection procedures that know how to compute each of V, I and R given the values of the other two parameters. We have also seen that objects are actually collections of variables, with constraints on those variables. We now show in detail how constraints are satisfied, i.e. how selection procedures assign values to variables.

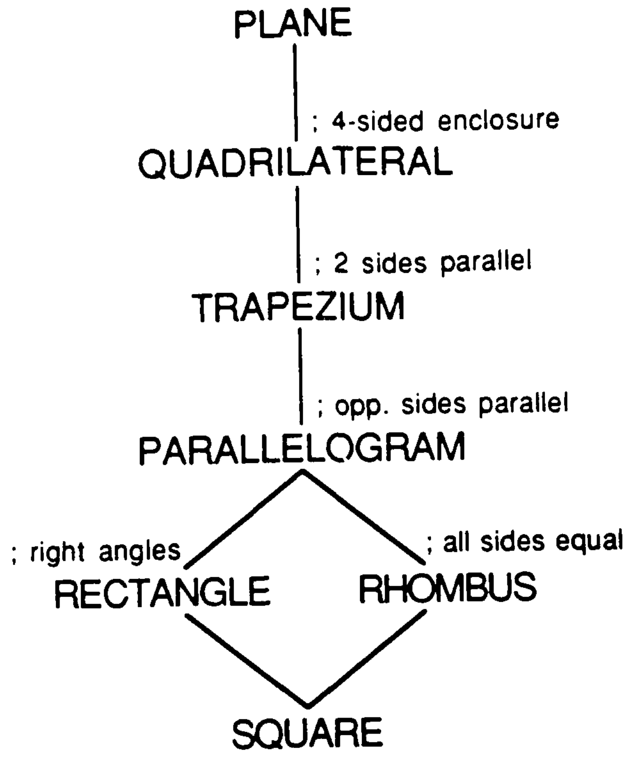


Figure 3-5: Inheriting Constraints.

Suppose we have an object describing a resistor. That object will contain values for the voltage, current and resistance of the particular resistor being described. Suppose that at some point, the voltage and resistance of that resistor become known; clearly the current through the resistor could be computed.

How is that computation initiated? Figure 3-6 shows the resistor object (object1), its variables, and some constraints (realized as selection procedures). When the object is asked to assign a value to a variable, it assigns that value and then it executes all the selection procedures having that variable as an input parameter. Those selection procedures are found using the arrows that emanate *from* the variables. Some of the executed selection procedures might decide that they have enough information to bind another variable. The target variable is found using the arcs that emanate from constraints to variables. If a selection procedure does indeed have enough information to bind a target variable, that procedure computes the value for that variable and asks the object to do the assigning. Once again, the object may decide more than just the requested variable.

Eventually the computation resulting from an update request will terminate. When that happens, the object considers that all the relevant deductions have been made, and that the object is in a consistent state.

It is possible that an object may be given inconsistent input parameters. This will be detected (sooner or later) as an attempt to assign two different values to the same

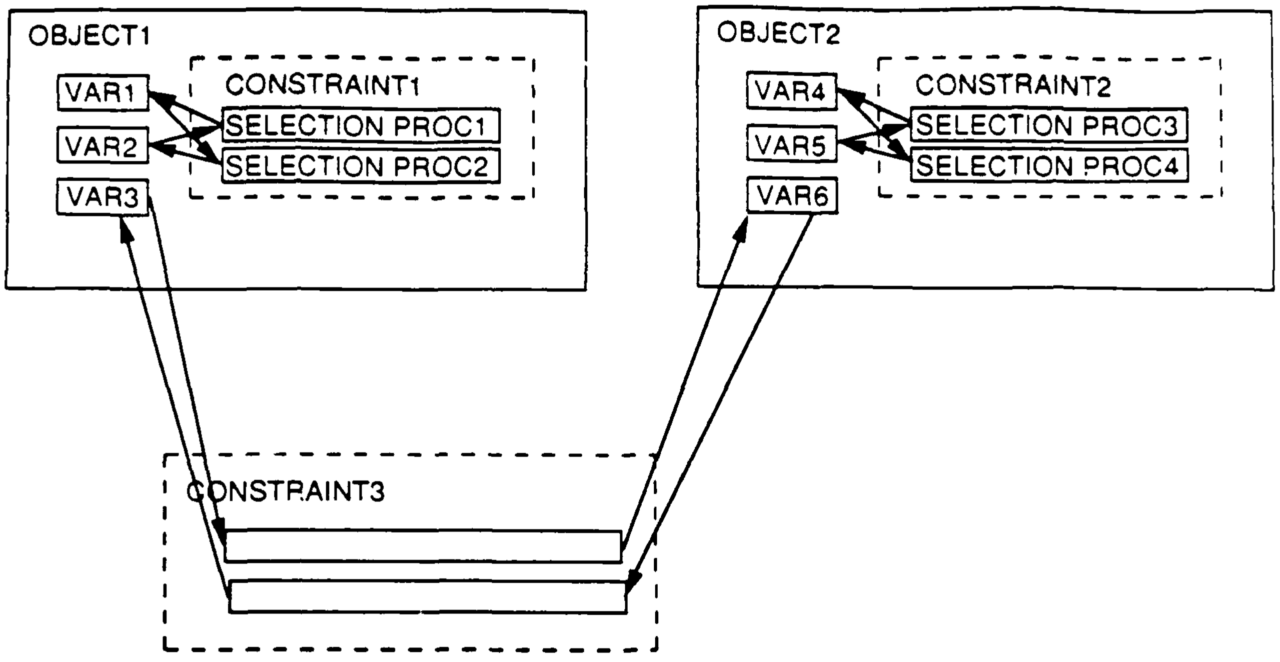


Figure 3-6: Satisfying Constraints.

variable. In such circumstances the constraint reasoning system must track down the roots of the inconsistency, and must reject one of the inconsistent inputs. This rejection takes the form of a retraction. The retraction may be entirely automatic, in which case the reasoning system randomly chooses which of the inconsistent inputs to retract; or it may be user-assisted, in which case the reasoning system will ask the user to identify the input to be retracted.

3.8.3 Interactive Constraint Choosing

So far we have focused on constraints that apply to the variables within an object; these are "intra-object" constraints. Examples of such constraints were the constraints that turned quadrilaterals into trapeziums, and trapeziums into parallelograms, and so on. There may also be constraints that apply to variables in different objects; these are "inter-object" constraints. An example of such a constraint is that the current flowing through each of several resistors in series is the same. In an image interpretation application, inter-object constraints may determine occlusion (which of two objects is hidden because further away) – among other things. The distinction between intra- and inter-object constraints carries over into image interpretation as a distinction between two kinds of questions. Intra-object constraints give rise to the question "*What* is this image object (a parallelogram or a rectangle?)" while inter-object constraints give rise to the question "*How* are these image objects spatially related (which occludes which)?"

The task of image interpretation is to decide which constraints are relevant to the image being interpreted. That is, image interpretation is a process of constructing a plausible theory about what is visible in an image. We claim that constraints are a convenient medium for expressing such a theory, because they allow us to say only what we believe without committing us to too much; and because they allow us to check that our theory is physically possible (by satisfying all the constraints).

For the moment we are proposing to construct our interpretations or "theories" interactively - a human will look at an image and will select the relevant constraints, and will inform the constraint reasoning system about what constraints apply where. We view this as a first step that allows us to gain experience with constraint-based image interpretation. In the long run it may be possible to have an image understanding system do (some of) the constraint selection for itself.

In summary, the constraint satisfaction techniques we shall utilize in our object models are based upon:

- **Simultaneous Algebraic Equations**

  Several constraints can be expressed as simultaneous algebraic equations. This is true for most geometrical relations under perspective projection or through direct determination with a depth sensor. For sets of linear equations, the solutions can be obtained directly. In this case, the constraint network is used to automatically determine which are the redundant variables. For sets of nonlinear equations, the situation is generally unsolvable, although there are constraint networks for solving sets of quadratic equations. The order of a variable in an equation can be used to determine whether we ascertain its value by sensing it directly or let it be determined via propagation in the constraint network.

  For constraints that can be expressed as algebraic equations, there are developed packages for solving them that we would interface to a constraint-based representation. This includes Mathematica [Wolfram - 88] and MACSYMA. We will also investigate interfaces to numerical packages for solving sets of simultaneous equations.

- **Predicate Logic Expressions**

  Expressions in the first order predicate logic can also be handled by constraint networks. This is necessary for assertions regarding propositions such as something being on top of something or attributes such as something being red. The solutions to the algebraic equations give us parameter value while the solutions to the simultaneous predicate logic expressions give us a consistent set of predicate or the determination that such is impossible. The underlying mechanism used for first order predicate logic constraints is similar to that used for simultaneous algebraic equations (rewrite rules) and there are also no guarantees of always finding an answer. There are a wide range of tools available for handling such constraint networks that we investigated including PROLOG and a Truth-Maintenance System (TMS).

- **Simulation Techniques**

  Simulation techniques are useful for representing effects such as gravity or the interaction of multiple objects in the world model.

4. Representation of Object Models

Object models are used to describe entities such as terrain patches, roads, cars, vegetation, and whatever else a telerobot can encounter in the environment that needs to be represented. Object models need to reflect the underlying physical properties of the world to make autonomous instantiation and refinement possible. When the human positions a tree along a line of sight and the system refines it, it must be sure that the tree is consistent with the current instantiations of the ground and of gravity. This information needs to be represented in such a fashion that the relationships between different classes of objects is made explicit so each object can not be treated as a special case. Our work focuses on achieving this by creating a general format for representing objects which is compatible with constraint representation and satisfaction techniques.

4.1 Recognition using Constraint-Based Object Models

The basic result of human directed recognition is to match existing object models to images and other sensory information. This process of matching is called **object instantiation**. It can be visualized as projecting wireframe displays corresponding to the object models onto images from the telerobots while the wireframes are aligned with the images of viewed objects. This produces information concerning object type, object attributes, position, and orientation, that enable actions. Our overall approach is based upon the combined use of structured a prior object information, focused use of perceptual strategies to direct and verify model matching, and constraint-based reasoning.

Our object models are implemented using constraint-based descriptions described in Section 3. This is a declarative representation in which all the attributes and relationships which are be true for an object are expressed as simultaneous constraints [Borning - 81, Borning - 79, Leler - 88, Mundy, Vrobel and Joynson - 89, Steele, Jr. - 80, Steele, Jr. and Sussman - 78, Lawton - 80]. This is computationally realized in a network [Steele, Jr. - 80, Mundy, Vrobel and Joynson - 89]. When a particular value or relation is determined, it will propagate through the network to set other values according to the specified constraints. Thus, in using constraints, instead of having to fill in each attribute or feature of a model to instantiate it, a small number of such attributes can be determined and then the constraints enable determining the value of other related attributes. This enables the system to function opportunistically because it can use any attributes to infer the other attributes of an object. Our use of multiple sensors will generate a very large number of simultaneous constraints. In addition, inconsistencies can be determined in the constraint propagation process. Constraint based representations allow for propagating information and inferences while maintaining unspecified hypotheses.

There are different types of constraints and associated propagation/satisfaction techniques. We use constraints which are expressed as predicates in the first order predicate calculus and also as sets of simultaneous second order polynomials which express geometrical relations in terms of distances and angles. Constraints express geometrical, interval, and logical relationships between parts of an object and the relations between a sensor and an object. In constraint networks, one type of node corresponds to variables and another type of node corresponds to the constraints between variables. Arcs describe the involvement of variables in constraints. Constraint propagation is the process by which determined variables are used to set undetermined variables by procedures attached to the constraint nodes. Example constraint predicates for our objects correspond to relations such as order, symmetry, connectivity, relative orientation, relative scale, coincidence (such as point to curve, curve to surface), and containment. Many of these predicates are topological in nature and invariant under perspective projection.

4.1.1 Recognition Processing

In instantiating an object model, three related constraint networks are formed:

1. The **Intrinsic Object Constraint Network** is for object properties and relationships which are independent of position and orientation with respect to other objects and sensors. Examples are such things as the relative size and orientation of components of an object and the contour determined by the intersection of object surfaces.

2. The **Coordinate System Mapping Constraint Network** describes the position and orientation of an object relative to another object or sensor. Because of sensor calibration in our system, the determination of these values for an object relative to one sensor will generally fix them relative to other sensors.

3. The **Sensor Feature Correspondence Constraint Network** describes the potential matches of extracted image features with corresponding components of an object model. A typical example is the correspondence between an image curvature junction and a point on a object model or between an image contour and the corresponding edge of the object.

These different networks are all expressed in terms of the same underlying constraint predicates and can be viewed as a single network in terms of constraint propagation. Thus perspective projection is expressed as a coincidence constraint between a ray of projection and a point feature. The constraints in these different networks are highly inter-related during recognition and are a major reason for the use of constraint-based representation. For example, an incorrect correspondence of sensor features to object model features will generally lead to an inconsistency in the determination of intrinsic object attributes. Instantiating an object involves accessing a generic constraint network corresponding to an object model and connecting it to constraint networks for coordinate system relations and sensor feature correspondences.

Recognition is performed as a search process wherein each node of the search space is an instantiated constraint network. Evaluation of a node is based upon how constraint propagation develops in a network, such as whether it cycles, develops an inconsistency, or has unspecified values. For these cases, the search process creates new nodes by adding or modifying values from the previous constraint network. This is done by directly sensing additional environmental information for variables which lead to an inconsistency or by establishing different image feature to object model correspondences.

Much of this search process is inherently parallel such as the constraint propagation within a search-node and the parallel evaluation of constraint networks which are based on different image to model feature correspondences. There are three major phases for model matching: 1) The initial extraction of perceptual information; 2) Forming initial correspondences between image features and object models features; 3) Position and attribute refinement with verification.

4.1.2 Sensor Feature to Model Feature Correspondence

This involves establishing an initial correspondence between extracted image features and model components. This correspondence determines several constraints such as which point features are constrained to lie along rays of projection, curves constrain object edges to lie along surfaces, and regions constrain object surfaces to lie within volumes.

There are general problems with establishing these correspondences autonomously. First, the image features may not be cleanly extracted even though they are present in the image: considerable processing can be required to find image features. Contours and regions can be fragmented and relations between features can require grouping processes. Secondly, there is combinatoric problem. There could be several potential matches between an extracted image feature and several different model features. In general, especially initially, due to ambiguous image to model feature correspondences and significant ambiguity in the uncertainty intervals associated with some of the constraint values, the propagation process will usually stop without having established a unique match. For this reason, it is good for the recognition search to establish several constraint networks with different correspondences early in the processing or, as in our case, for the human to specify a small number (generally 2 or 3) of correspondences prior to autonomous recognition. It should be noted that the required number of correspondences is not extensive, especially when the object models are rigid with known scale or orientation. Three points can fix the position and orientation of an rigid object model in three space.

There are several heuristics to guide the determination of correspondences. Associated with each object model feature and constraint is a description of the type of perceptual feature that can be seen for it and also the relative viewing direction from which it can be seen. Features are ranked according to their perceptual salience which reflects how viewpoint independent the feature is and how unique it is with respect to the object and to other objects. During recognition, an agenda is maintained of pending queries for finding features.

4.1.3 Position and Attribute Refinement and Verification

This involves determining the position of the model features in three space given the correspondences by solving a set of algebraic equations in which the multiple constraints are used to reduce the number of unknowns. In addition, this phase involves verifying the correspondence of features based upon the predicted appearance of image features, either by direct sensing or by queries over the perceptual structure data base in the associated image areas. The constraint propagation routines associated with the nodes in the sensor correspondence network perform perceptual actions to verify the predicted features.

A key concept is the utilization of perceptual strategies to acquire environmental information when the constraint process gets stuck. Such perceptual processing will often involve the system directing it's sensors to acquire particular information. Such controlled use of sensors restricts the number of underdetermined variables in an inference process [Aloimonos and Swain - 85]. Examples are focused depth determination by keeping multiple cameras focused on a point while moving them for triangulation with simplified correspondence or looking for a particular feature by moving a sensor towards a determined location convolving directly with a mask corresponding to the anticipated feature or projecting a light grid onto the portion of the environment to infer the occurrence of discontinuities, depth, and surface orientation. These are important sources of information to unlock constraint propagation or resolve an inconsistency without necessarily alerting the human controller. We envision a library of such routines for obtaining environmental information, either using a directable range sensor or multi-camera stereo. Associated with each routine is some cost estimate for performing it and also the corresponding type of attribute and constraint that it can infer. Examples are given in Section 5.

4.2 General Object Representation

The general format for object models consists of (Figure 4-1):

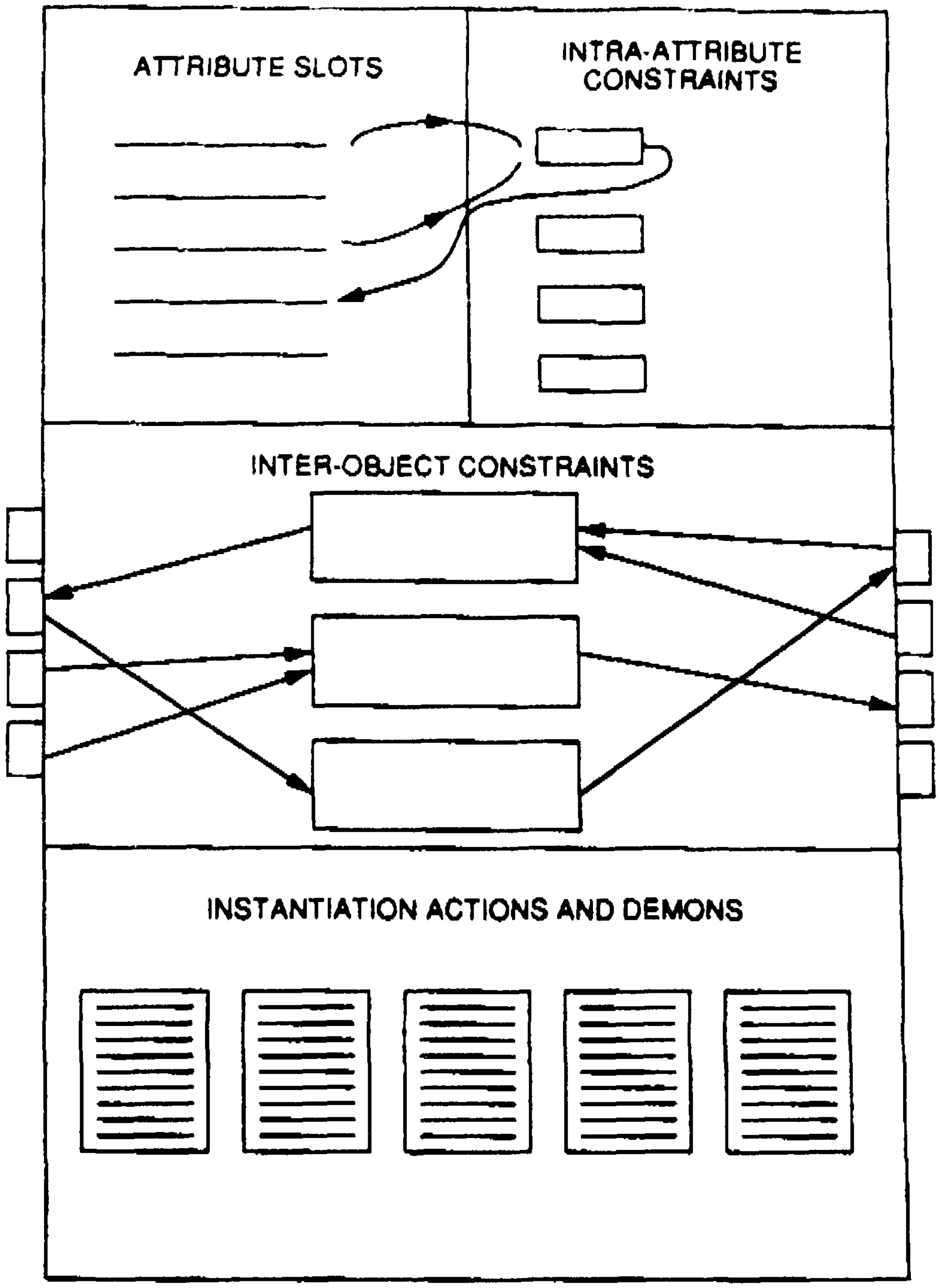


Figure 4-1: General Object Model

- **Attribute Slots** which describe the different components of an object. For example, a line-segment has attributes such it's length and it's endpoints. Attribute slots can be filled by variables from classes such as numbers, vectors, items from an enumerated list, and pointers to other objects.

- **Intra-Attribute Constraints** describe the relations between different attributes as a part of the characteristics of an object. For example, the length of a line segment can be determined from it's endpoints or one endpoint can be determined from the orientation of the line, it's length, and the other endpoint. These relationships are stored in the intra-attribute constraints. There are different forms of these constraints based upon the different type of constraint representation that is used. In one form, they are expressed as algebraic equations which are compiled into the constraint network which corresponds to the world model when the object is instantiated; or they can be a set of first order, predicate expressions that describe the relationship between the attributes. These predicates are added to a predicate calculus constraint network in the world model when the object is instantiated; or it can be a procedure that describes how to determine the values of one attribute slot given the others. A single attribute may have constraints specified only for itself in terms of bounds that it must always be limited by or that it's value must remain constant

- **Inter-Object Constraints** are similar to Intra-attribute constraints, except they are used to described relations between different objects in the world model. For example, an inter-object constraint would express that a particular type of object needs to be aligned with gravity or that it must be in contact with the ground plane if it is a terrestrial object. These in general involve checking through the current state of the world model.

- **Instantiation Action and Demons** are procedures which are executed when objects are instantiated in the world model. Demons associated with an object describe actions to take when some change occurs in the world model.

4.2.1 Use of Object Oriented Programming Methodology

To realize our object models, we have used an object-oriented programming methodology as supported by languages like CLOS [Bobrow et al. - 87], Object LISP, Flavours, or C++. In object-oriented programming [Cox - 84] data structures and their associated procedures are combined into integrated units called **objects**. A very simple instance of object oriented programming in conventional programming languages is found in the notion of **type coercion**. A programmer can specify the operation of addition for numbers with out having to specify the required particular type of number being added, even though the internal representation and operations can be quite different. In object-oriented programming this is generalized so that when an instance of an object is operated upon, the correct procedure to perform that operation is determined by the object. Thus it is possible to have several different types of objects which can implement the same abstract operation in ways specific to it.

Another important attribute of object-oriented programming is inheritance. New objects can be defined in terms of other objects by the specialization of their associated procedures and data structures. This leads to considerable abstraction in programming. Inheritance applies to both the data and procedural aspects of objects, so the actions associated with a defined class of object can be modified (or shadowed) for more specific instances. In some object-oriented programming environments, inheritance is very general and objects are allowed to inherit from multiple classes of objects. This allows for expressing general classes of objects and also being able to define specific instances which correspond to exceptions.

The effect of the use of objects is to significantly increase modularity, flexibility, and code maintainability. From the perspective of a developer, this allows for rapid prototyping (studies have indicated by as much as 300 percent [Schmucker - 86]). From the perspective of a naive user, the objects correspond directly to natural units for describing a domain. An object can be viewed as an agent that uses its associated processes to respond to some condition, such as the state of another object or a globally accessible resource. Thus, a user can point a mouse at an object, click on it, and the object will respond in an appropriate manner without having a large number of parameters specified.

Object oriented techniques allow us to represent the explicit inheritance relations between objects. The inheritance relation can be applied to all aspects of an object, including constraints, attributes, and attached demons.

We develop three basic types of primitive objects: geometrical, material, and constraint/relationship. These are to be combined to create general terrestrial objects.

4.3 Primitive Objects

We distinguish two different types of object models. **Primitive Objects** correspond to very basic attributes and relations that serve as building blocks to construct more complex objects. **Terrestrial Objects** correspond to entities and relationships which are commonly found in the world, such as gravity, ground, and trees.

There are three classes of primitive objects: **Geometrical Objects** which are used to describe shapes; **Material Objects** which are used to describe the stuff that objects are made out of; and **Constraint/Relationship Objects** which describe spatial and logical relationships between the components of objects and constraints on their attributes.

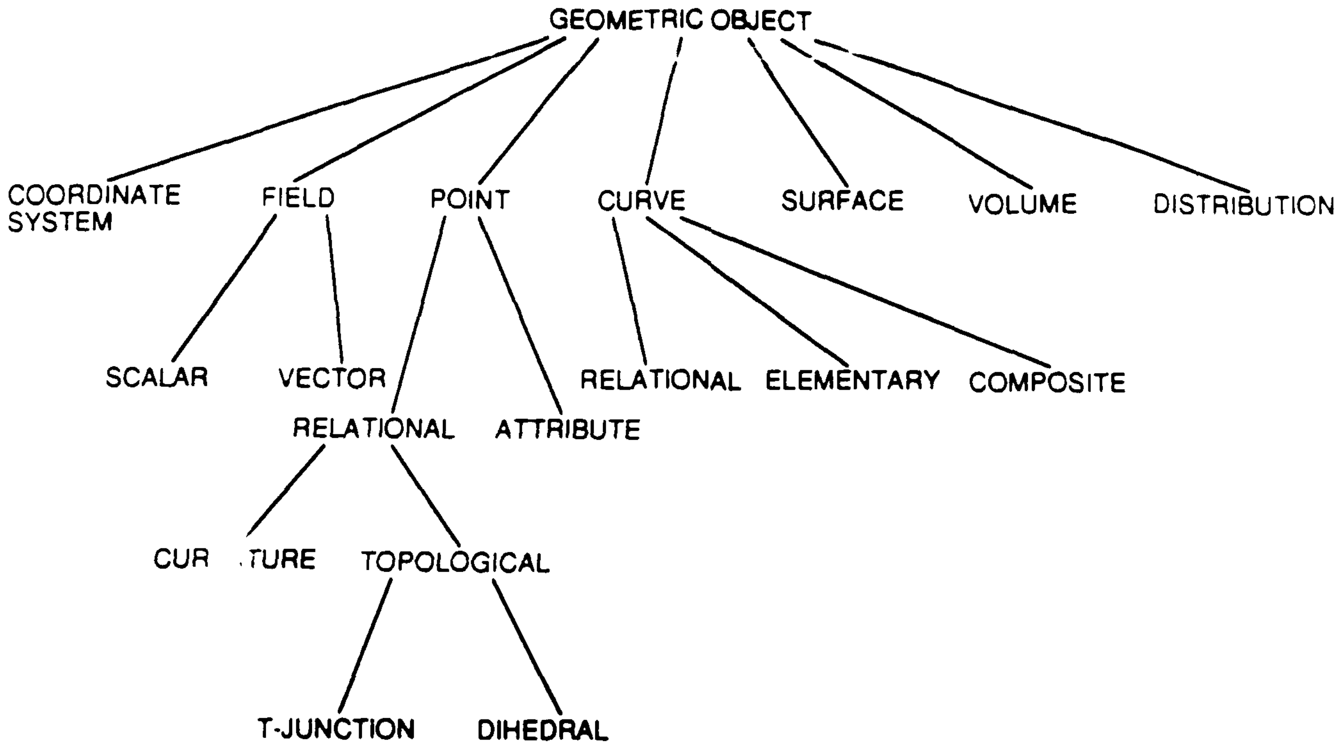


Figure 4-2: Geometric Object

4.3.1 Geometrical Objects

The inheritance hierarchy for geometrical objects is shown in Figure 4-2. A coordinate system is a geometrical object which has it's conventional meaning in terms of number of dimensions and the various types (cartesian, polar, cylindrical). Almost all the geometrical objects have an associated coordinate system. A **field** corresponds to a scalar, vector, or tensor fields as used in physics. Thus, gravity is defined as a constant vector field with an associated coordinate system.

Point, curve, surface and volume objects also correspond to their typical usages and reflect 1, 2, and 3 dimensional objects. Each is specialized into similar types. **Relational** refers to a geometrical object which is defined by the relationships of ‹,ther geometrical objects, typically through intersection. Thus, a relational-point object is used to describe the intersection of curves; and a relational-curve is used to describe the intersection of surfaces. The relational class is generally subdivided into topological to refer to non-metric properties such as connectivity. **Elementary** refers to particular types of curves, :urfaces, and volumes. Thus, a straight-line, a circle, an ellipse, and segments of these, are elementary types of curves. **Composite** refers to an object which is made up of elementary objects of the same type. Thus a sequence of straight line segments constitutes a composite curve.

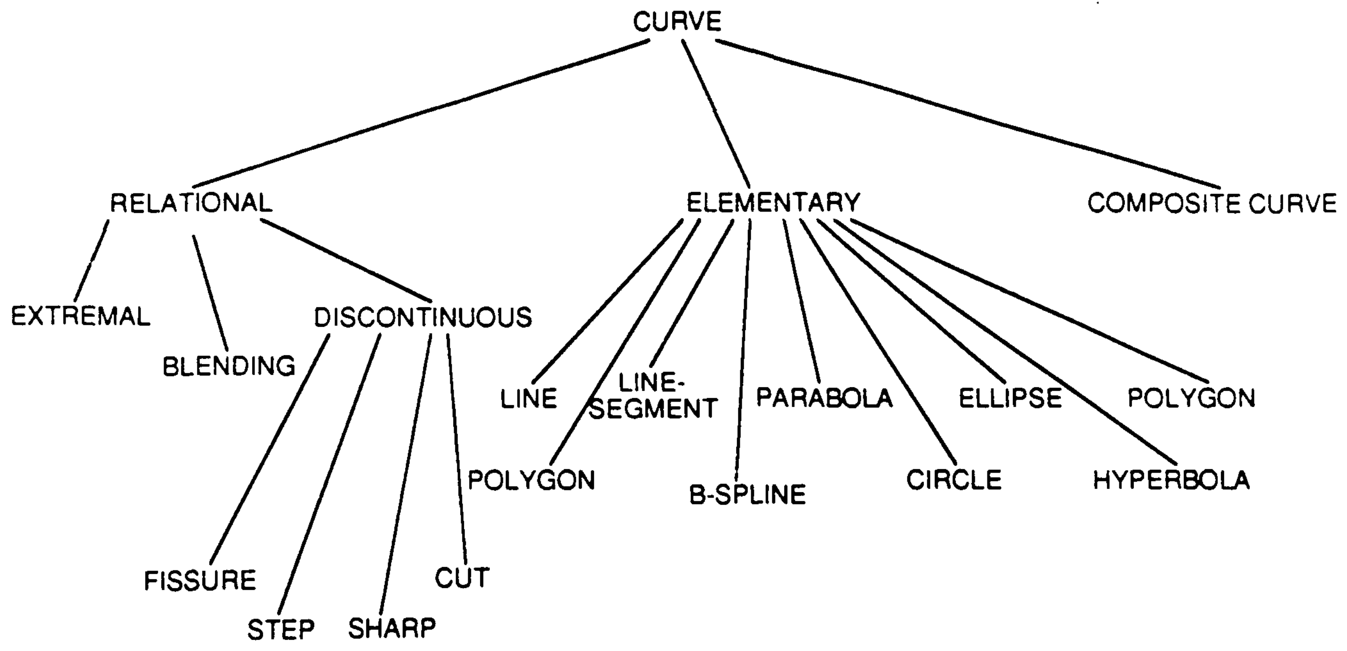


Figure 4-3: Curve

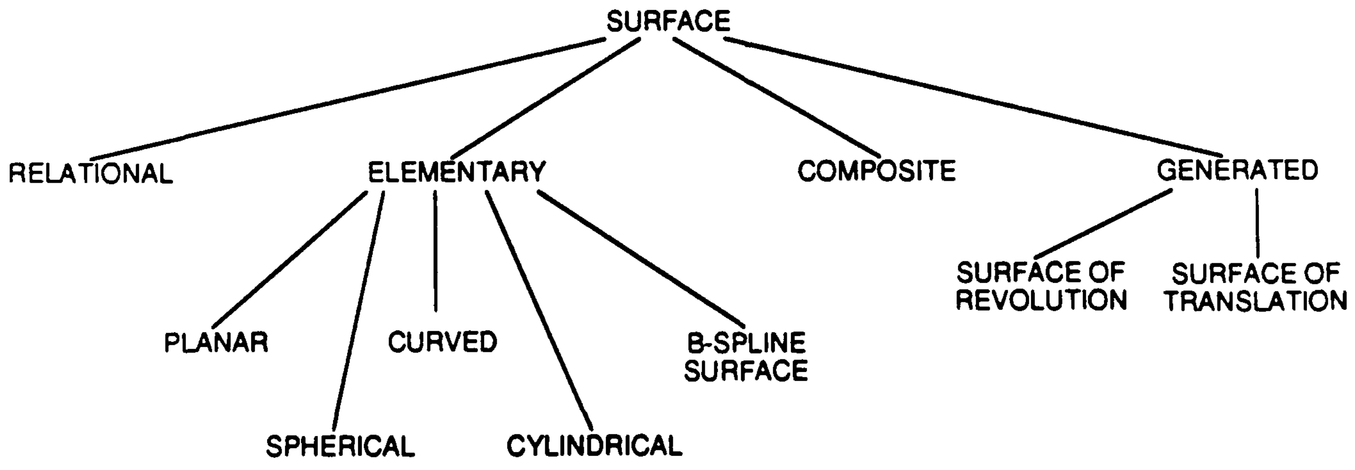


Figure 4-4: Surface

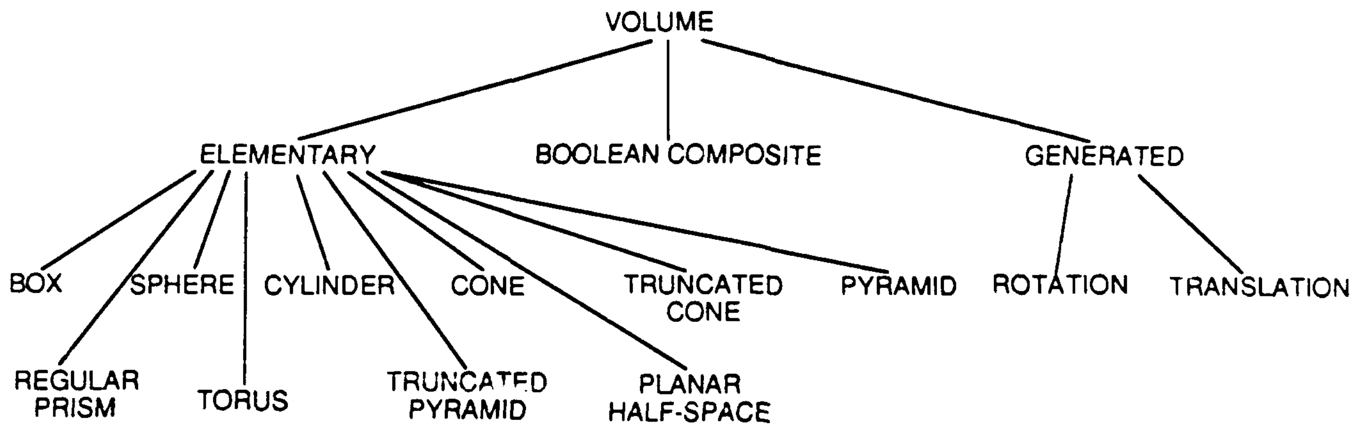


Figure 4-5: Volume

Distributions are used to describe *objects such as tree-crowns*, vegetation patches, and land-masses. These are rough shapes which can be described parametrically using a probability distribution.

4.3.2 Material Objects

Material objects are used to describe the basic stuff which makes objects up. Ultimately, these properties correspond to the underlying atomic properties of the substances which make up the world, such as concrete, asphalt, copper. From material science [Cotterill - 85] there are well established physical properties and classifications for substances. For perceptual utility, it is useful to have rougher classifications in terms of more general objects such as metallic, vegetation, water, and so forth. There are clear inheritance hierarchies between different material objects.

The general class of material objects is distinguished by a small number of attributes [Gibson - 86]:

- Hardness - how rigid a type of substance is. This is described by a scalar normalized between 0 and 1.
- Viscosity - the resistance to flow of a substance. Also described by a normalized scalar.

- Density - the mass per unit volume of a substance. Described by a normalized scalar.
- Coh^esiv^ness - the strength of resistance to breaking of a substance. Described by a normalized scalar.
- Elasticity - the tendency to regain the previous shape after deformation. Described by a normalized scalar.
- Plasticity - the tendency to held the previous shape after deformation. Described by a normalized scalar.
- Chemical reactivity - a vector describing the susceptibility of a substance to chemical reactions such as solubility in water and air.
- Characteristic Spectral Reflectance - how the substance absorbs and reflects light. This is described by a characteristic function from wavelength to the amount of reflected energy.

We are treating substances as though there are uniform. This is not really the case. Objects can be described in terms of multiple levels: such as a cell, a leaf-vein, a leaf, a branch, a tree, a forest. Understanding the world doesn't require that all possible levels for an object are used to interpret it, but that the appropriate one is used, based upon the objectives of the system and what c.. be perceptually determined. Thus, at one distance, the tree crown is a single object made up of general vegetative material, while closer, it consists of branches and leaves. To deal with this, there is also a nesting attribute associated with material objects to describe at what distances what are the appropriate characteristic objects to describe the material. This is a table which describes the components of the objects indexed by scale. The perceptual processes can use this, coupled with the resolution and determined depth of object to determine the correct scale to use. Thus we add:

- Nesting
  - Scale/distance related to level of description
  - Characteristic pattern for perceptual processing specified as a group (see Section 5)

4.3.3 Constraint/Relationship Objects

The constraint objects deal with relationships such as rigidity, coincidence, relative orientation, absolute scale, order, containment, rigid body transformation, and symmetry (rotational, reflectional, radial, bilateral).

For example, perspective projection is defined as a coincidence relation between an environmental point and a ray of projection (see Figure 4-6). This is expressed by the equation relating an image point to the corresponding environmental point:

$$P_{ei} = z_{ei} I_{ei} \tag{4.1}$$

Thus, when there is a perspective constraint, there is a relation between an image point, and environmental point, and a parameter describing environmental depth. The perspective constraint object is able to determine any one of these values given the other two. This shown in Figure 4-7 where there is a perspective constraint between the environmental point P_{ei} and the image point I_{ei}. These are represented in a partitioned graph where one side corresponds to the intrinsic attributes and the other to the image-to-feature mapping constraints. When there are many points visible in an image, there is a set of such constraint objects for each point.

In general there will be several other relations between points such as known orientation, known distance or rigidity. For example, known distance is expressed by under perspective:

$$\begin{aligned} D^2 &= z_{ei}^2 (I_{ei} \cdot I_{ei}) \\ &\quad + 2 z_{ei} zej (I_{ei} \cdot I_{ej}) \\ &\quad + z_{ej}^2 (I_{ej} \cdot I_{ej}) \end{aligned} \tag{4.2}$$

and known orientation in the environment by

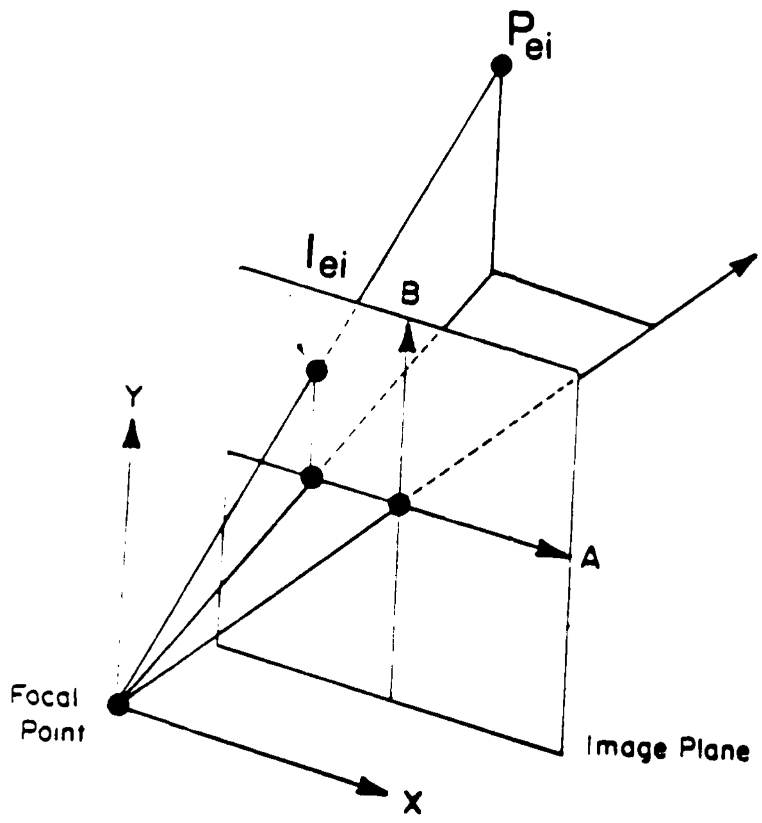
$$\frac{(P_{ei} - P_{ej}) \cdot \overrightarrow{V}}{\| \overrightarrow{V} \| \| P_{ei} - P_{ej} \|} = n \tag{4.3}$$

Interestingly, for example, while these constraints can be expressed in a network and handled by direct propagation (see Figure 4-8), simultaneous constraints often will share terms and can be simplified. This will need to be explicitly recognized. In particular, known orientation and distance between two points yields:

$$P_{ei} - P_{ej} = \overrightarrow{V} \tag{4.4}$$

$$\begin{gathered} z_{ei} I_{ei} - z_{ej} I_{ej} = \overrightarrow{V} = (v_1, v_2, v_3) \\ z_{ei}(a_{ei}, b_{ei}, 1) - z_{ej}(a_{ej}, b_{ej}, 1) = (v_1, v_2, v_3) \end{gathered} \tag{4.5}$$

which is an overconstrained set of simultaneous linear equations which is directly solvable.



$$P_{ei} = (x_{ei}, y_{ei}, z_{ei})$$

$$I_{ei} = (a_{ei}, b_{ei}, 1)$$

$$I_{ei} = (\frac{x_{ei}}{z_{ei}}, \frac{y_{ei}}{z_{ei}}, 1)$$

$$P_{ei} = z_{ei} I_{ei}$$

Figure 4-6: Perspective

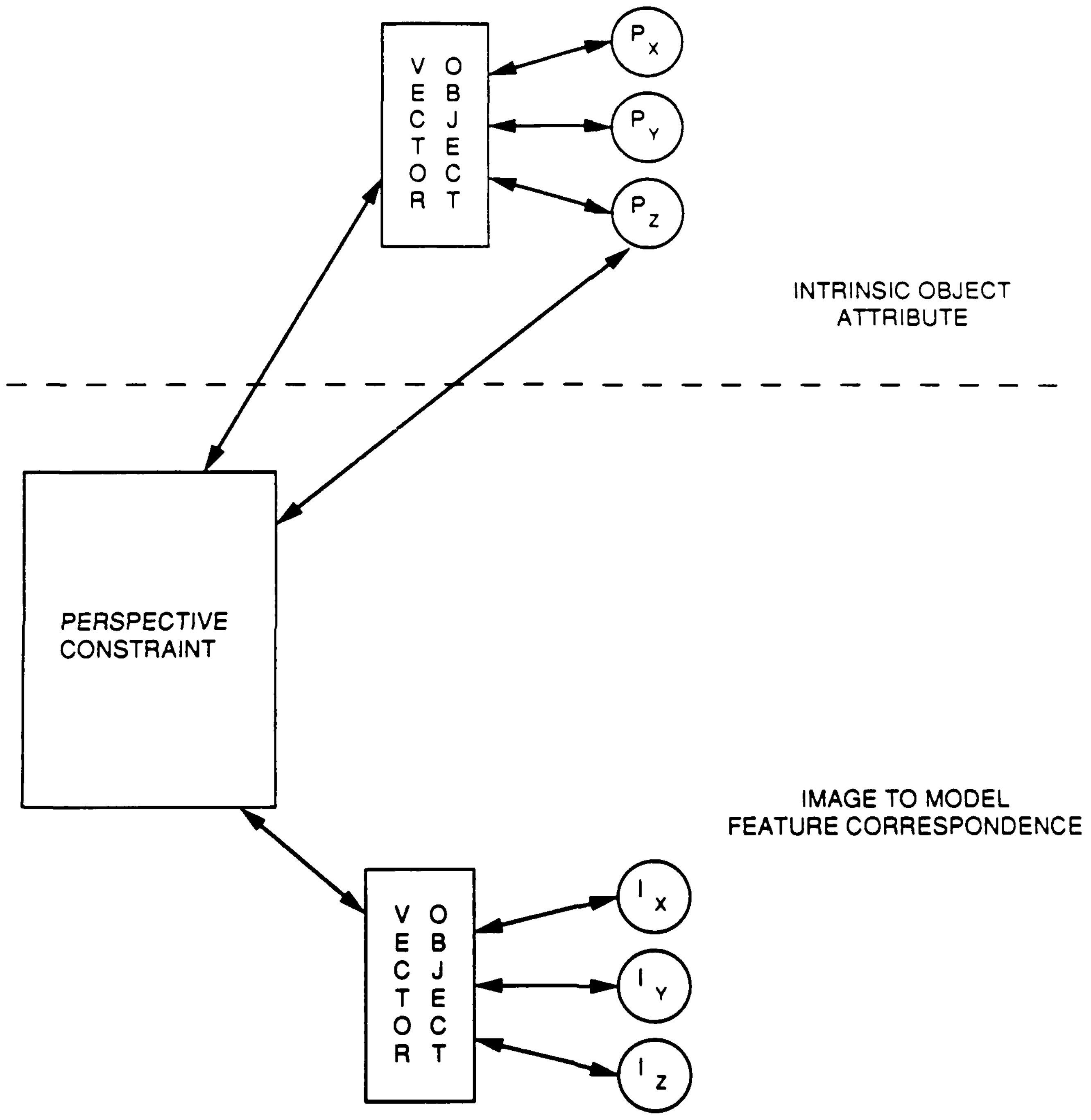


Figure 4-7: Perspective Constraint Network

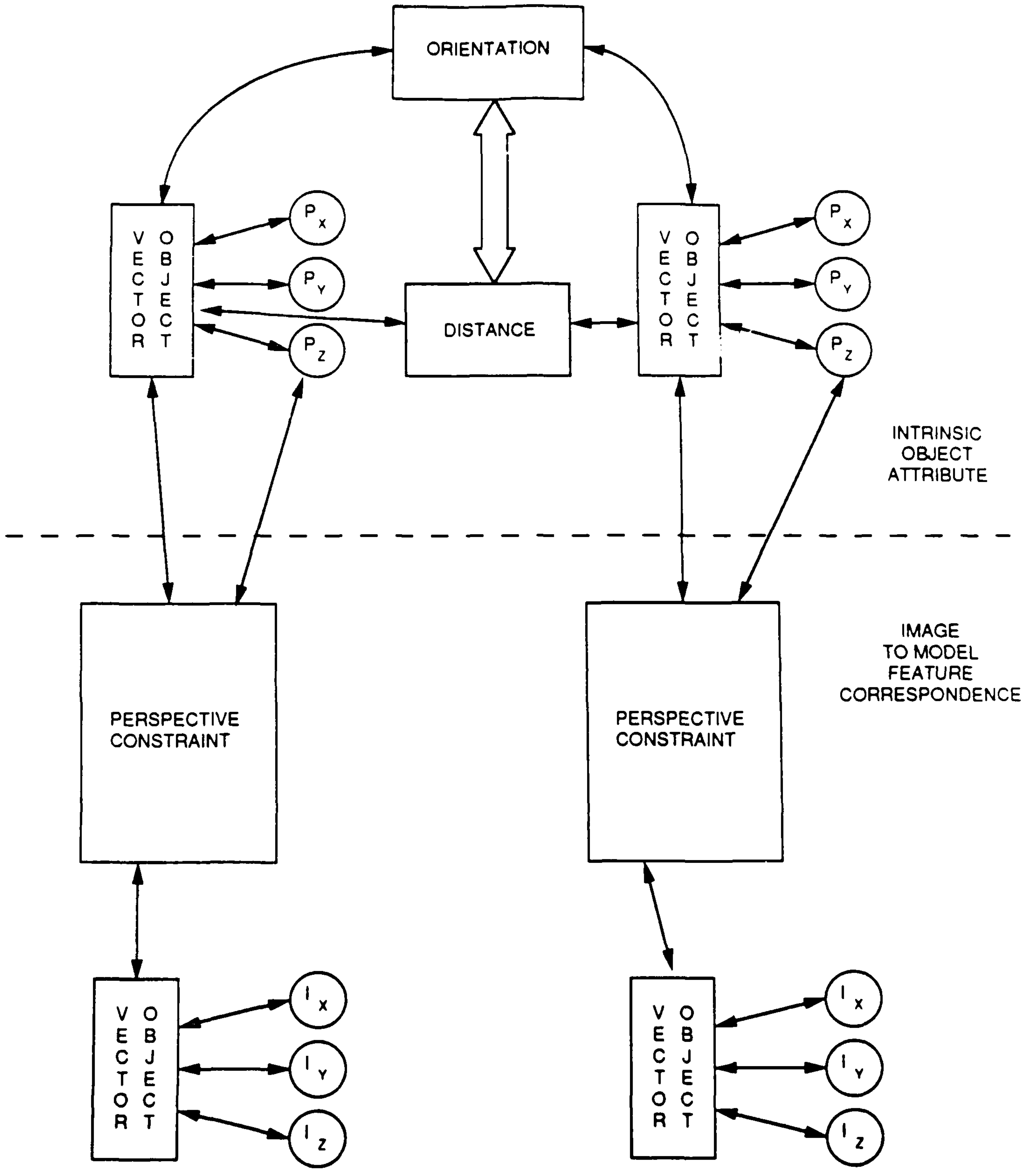


Figure 4-8: Simultaneous Constraint Objects

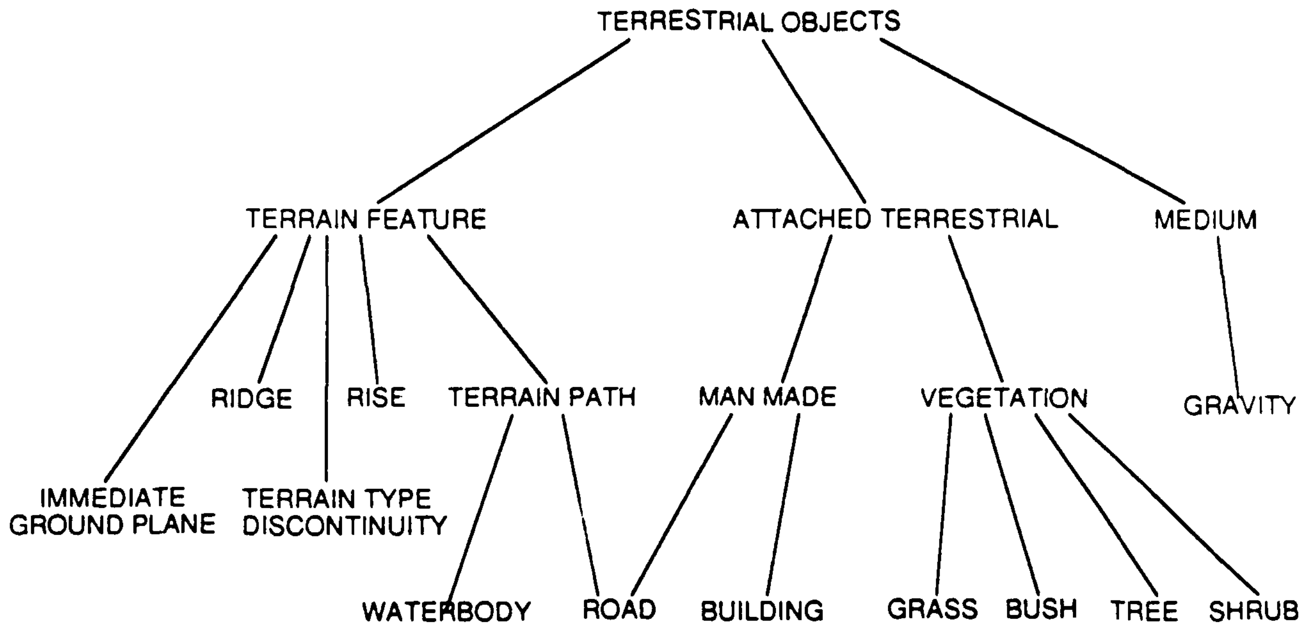


Figure 4-9: Terrestrial Objects

4.4 Terrestrial Objects

Terrestrial objects correspond to things found in the world. Figure 4-9 shows a simple hierarchy for terrestrial objects and Figures 4-10 through 4-12 show some we have constructed and used in the processing examples in Section 7.

- SLOTS
  - o COORDINATE SYSTEM (POINTER TO A TYPE OF COORDINATE SYSTEM)
  - o THE IMAGING SURFACE (POINTER TO A TYPE OF GEOMETRIC SURFACE)
  - o FOCAL LENGTH (SCALAR FOR THE DISTANCE FROM THE CAMERA. COORDINATE-SYSTEM. ORIGIN AND THE IMAGING SURFACE)
  - o LABEL-PLANE DEPTH BUFFER
  - o LIST OF VISIBLE OBJECTS IN WORLD MODEL
- INTRA-ATTRIBUTE CONSTRAINTS
- INTER-OBJECT CONSTRAINTS (THIS INCLUDES DEMONS THAT ARE PERFORMED: THINGS TO CHECK FOR WHEN A PARTICULAR TYPE OF OBJECT IS INSTANTIATED. THIS NEEDS TO BE UPDATED WHENEVER SOMETHING CHANGES, LIKE A NEW OBJECT ENTERS THE FIELD OF VIEW)
  - o DETERMINE VISIBILITY OF OBJECTS IN THE WORLD MODEL
  - o MAPPING RELATIONS BETWEEN CAMERA COORDINATE SYSTEM AND OBJECT

Figure 4-10: Camera

- SLOTS
    - o COORDINATE SYSTEM
    - o CONSTANT 3D VECTOR FIELD
- INTRA-ATTRIBUTE CONSTRAINTS
- INTER-OBJECT CONSTRAINTS
    - o FOR ATTACHED CAMERAS: CREATE THE 2D PROJECTED VECTOR FIELD IN REGISTER WITH THE SENSOR SURFACE; THIS IS THE PROJECTED-GRAVITY-FIELD
    - o FOR OBJECTS THAT NEED TO BE ALIGNED WITH GRAVITY: CHECK IF THEY ARE WITHIN SOME THRESHOLD THETA WITH RESPECT TO THE INSTANTIATED GRAVITY FIELD
    - o CHECK FOR CONSISTENCY BETWEEN THE TRANSFORMATION RELATING DIFFERENT CAMERAS AND THEIR PROJECTED GRAVITY FIELDS
    - o FOR OBJECTS THAT NEED TO HAVE TRANSITIVE CONTACT WITH RESPECT TO THE DIRECTION OF GRAVITY: POINTS ALONG OBJECTS WHICH HAVE DEPTH DISCONTINUITY WITH RESPECT TO THE PROJECTED DIRECTION OF GRAVITY

Figure 4-11: Gravity

- SLOTS
  - o COORDINATE SYSTEM
  - o NORMAL RELATIVE TO COORDINATE SYSTEM
  - o EQUATION OF PLANE
  - o POINTS ON THE GROUND PLANE
- INTRA-ATTRIBUTE CONSTRAINTS
  - o RELATION BETWEEN POINTS ON THE GROUND PLANE AND THE NORMAL
- INTER-OBJECT CONSTRAINS
  - o FOR EACH INSTANTIATED CAMERA IN THE WORLD MODEL:
    - — CREATE A HORIZON LINE OBJECT SPECIFIED BY THE RELATION BETWEEN THE CAMERA AND THE GROUND PLANE
    - — CREATE THE PROJECTED DEPTH MAP OF IMMEDIATE GROUND-PLANE POSITIONS AS A VECTOR FIELD IN REGISTER WITH THE IMAGE (IN REGISTER MEANS THERE IS A VALUE FOR EACH PIXEL IN THE IMAGE
    - — GENERATE THE PROJECTED RELATIVE RADIAL DIRECTION IN THE GROUND PLANE TO ORDER EXTRACTED IMAGE OBJECTS INDEXED BY A RADIAL COORDINATE (THETA)

Figure 4-12: Immediate Ground Plane

5. Perceptual Processing

Perceptual processing concerns how the telerobot uses it's onboard sensors to obtain information about the environment to refine the instantiation of object models. For effective functioning, it is critical that the perceptual processing be directed and constrained. either by the human controller or by the constraints provided by object models. We can distinguish three classes of sensors:

1. Direct Environmental Sensors – These yield such things as temperature, time of day, direction of gravity, and orientation of immediate ground plane. This information is obtained with little or no processing and acts to parameterize other perceptual processing, such as using the ambient temperature to set parameters on processing infrared images

2. Imaging Sensors – These form images of the surrounding environment in different spectral bands. These are passive sensors and using them to obtain environmental information can be computationally expensive. The sensors themselves are generally inexpensive, but need precision computer control of zoom, pan, and focus. The processing using such sensors tends to be computationally expensive, especially for single cameras. Multiple imaging sensors, if they are appropriately registered can yield precise depth information using stereo and variants of it.

3. Range Sensors– These obtain environmental depth information directly. A scanning laser range sensor forms an image of depth values. A pointing laser range finder determines environmental depth in a particular direction. In Section 2 we described a sensor which boresights a pointing laser range finder with a camera directly. These are active sensors which can be quite expensive but will yield reliable depth information with no computation.

Associated with each sensor will be multiple coordinate systems describing its position and orientation relative to the world model, instantiated objects, and other sensors. It is extremely important that all the sensors be registered so inferred information from one sensor can be used with respect to information obtained from another one.

Utilization of human directed and constrained perceptual processing is extremely important because of the current limited capabilities of autonomous perceptual processing in unconstrained natural environments. While a human may clearly see the boundary of a terrain patch from a water body, undirected segmentation processes will almost always not extract it. These constraints involve such things as limiting the extraction of particular types of image features to particular areas of an image determined by a human and directing depth sensors to particular locations.

The critical ideas in this section are representing perceptual information in a spatially tagged data base and expressing perceptual processing and recognition procedures as direct queries or search-based **grouping** processes which can be applied to this data base. This data base is called the **Perceptual Structure Data Base (PSDB)** and contains extracted points, contours, regions and other perceptual objects. It is described in Section 5.1. The grouping processes which operate over the PSDB find sets of related objects which correspond to specific types of patterns. An example is linking several disconnected edge fragments together to form an object's boundary or merging several regions together to form a predicted terrain patch. These are described in Section 5.2. Section 5.3 described perceptual processing routines for obtaining environmental information for direct incorporation into the constraint network for sensor feature correspondence. Much of the work in this section was developed as part of the Autonomous Land Vehicle project [Lawton et al. - 89] and was concerned with how to extract predicted terrain features derived from an a prior terrain map.

5.1 Perceptual Structure DataBase

The Perceptual Structure DataBase (PSDB) contains the images, objects, and groups that have been generated by sensors and perceptual processing (see Figure 5-1). Images are obtained from sensors or the results of low-level image processing. Perceptual objects are spatially indexed (with respect to an image) symbolic representations of such things as points, curves, regions, surfaces and volumes. **Groups** are sets of related objects in the PSDB that generally correspond to some type of significant pattern. Groups partition the perceptual structure database into related sets of objects. These relations can be a complicated hypothesized structure, such as a set of lines forming a particular curve or related image structures that are matched over a sequence of images. We stretch the concept of a group a bit to include descriptions of collective attributes of a set of objects for histograms and other data structures. The group structure is used to associate sets of images such as color groups, vector groups, pyramids, and motion sequences.

All PSDB objects have a small set of basic characteristics common to all objects of that type. In addition, arbitrary features can be associated with objects. In this way, there is a common representation that is extended as needed for specific processing. This is shown for the example of a curve object in Figure 5-2 where the attributes keyed by small letters represent the basic attributes of a curve and those in capital letters represent associated attributes.

Structures in the PSDB are accessed by a uniform query mechanism built around **filter functions**. A filter function takes a list of objects as its first parameter and, optionally, some additional parameters. The function produces a list of objects as its result. The function itself can range from simple attribute checking to a complex search or pattern matching operation. The objects returned are usually a subset

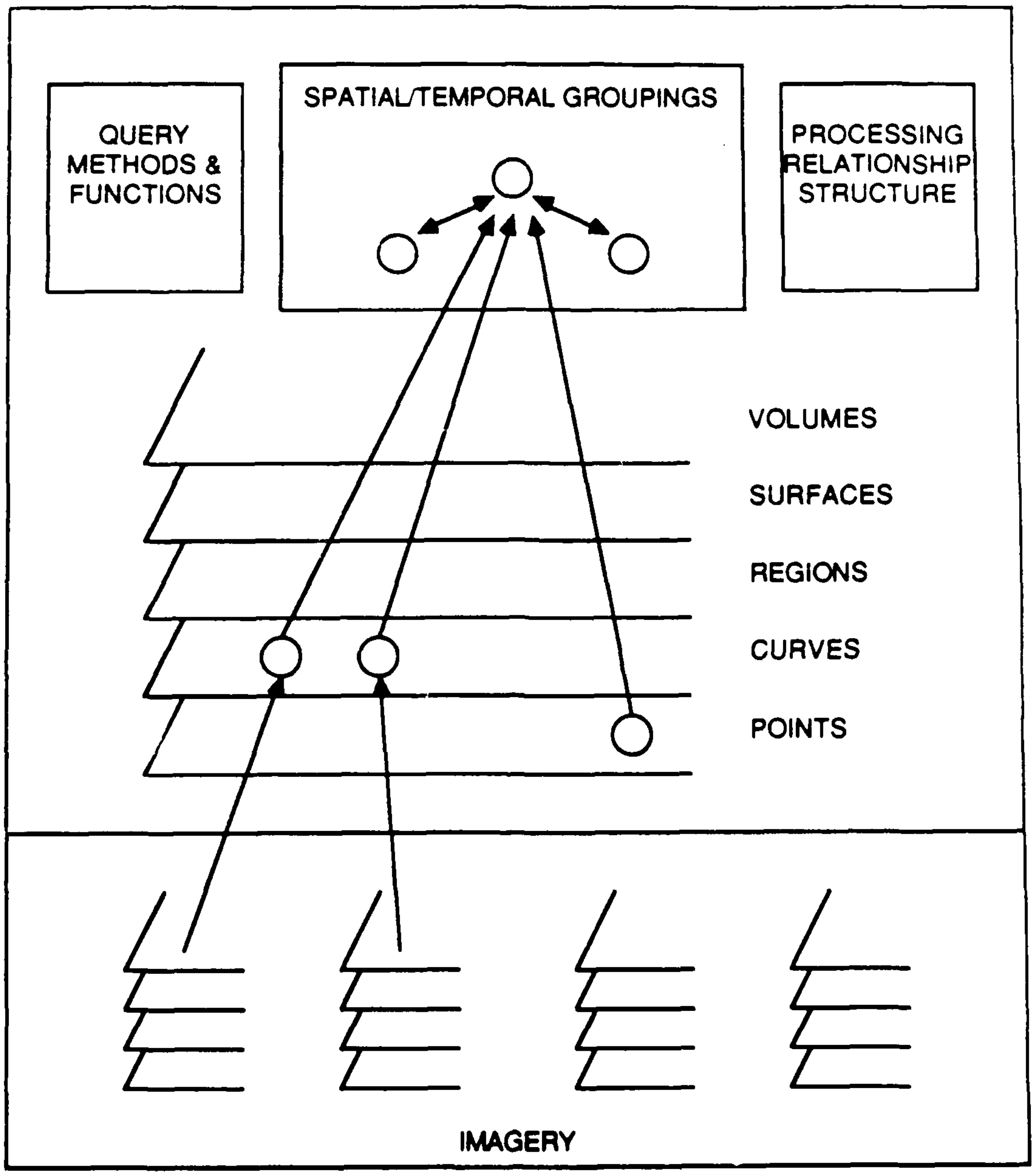

QUERY METHODS & FUNCTIONS
SPATIAL/TEMPORAL GROUPINGS
PROCESSING RELATIONSHIP STRUCTURE
VOLUMES
SURFACES
REGIONS
CURVES
POINTS
IMAGERY

Figure 5-1: Perceptual Structures Database

```
Curve  #<CURVE 175806283>
Point1  (205  291)
Point2: (274  285)
Length  81
Grid  6
Points  (205  291) (206  292) (207  293)        (276  287) (275  286) (274  285)
AVG-INTENSITY  159 27315
SUCCESSIVE-GRADIENT-DIFFERENCE   1 7660371   1 4782858   1 0026073
                                 0 5625856   0 6442404   0 7240415
GRADIENT  (-2.8555908  3 1515503) (-4 0343933  1 8386173) (-5.415619  1 3096824)
          (-0 03842163  -2.917389) (0 56552124  -2.6831152) (1 0851135  -2 1888733)
CONTRAST  5 7423206
AVG-GRADIENT  -5.7336416  0 31559697
LDC  (#<CURVE  175605520>
      (#<CURVE  175327156>)
      (#<CURVE  175505625>
         (#<CURVE  175327163>)
         (#<CURVE  175505632>
            (#<CURVE  175327170>)
            (#<CURVE  175327175>))))
```

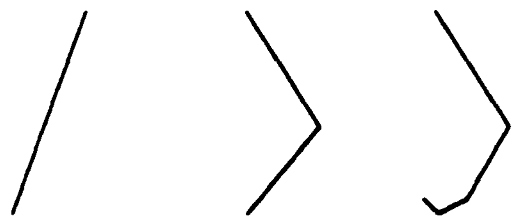


Figure 5-2: Instance of Curve Object

of the original objects, but can be a superset, or even a set of completely different objects. The filter functions are combined by using the filter constructor. The filter constructor takes a specification in terms of logical operators (AND, OR and NOT) and filter functions and generates the bindings and additional functions required to execute the query. This approach has proven to be a powerful and flexible one. There are a number of generic filter functions that are used with many different kinds of objects; we have written a large number of special purpose filter functions as well.

A particularly important type of image is a *label plane.* These are images that have a vector of indices at each location that point to all of the perceptual objects that have been extracted from an image at that location (see Figure 5-3). The label plane is very useful for relating objects extracted by different processes and at different resolutions. In addition, it provides a way of doing image-based geometric perceptual processing that is necessary for grouping operations. Filter functions can also refer to attributes of objects referred to in a label plane for finding all objects that have certain spatial properties such as extent and location (Figure 5-4). The analogy of a label plane for dimensions higher than two is straightforward, but computationally

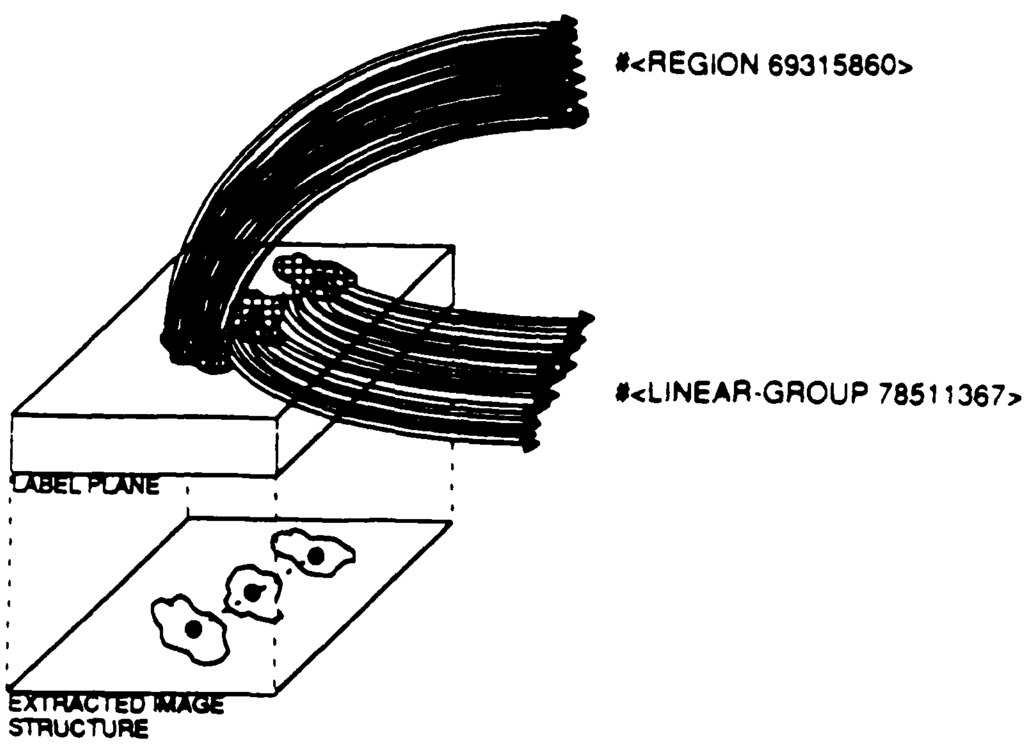


Figure 5-3: Label Plane Containing a Curve

expensive. Such volumetric space representations will require architectures such as the Connection Machine [Hillis - 85] or the CAAPP [Weems and Levitan - 87].

All perceptual objects have a TYPE attribute to indicate the object subclass. An object subclass indicates that some specific set of features has been calculated for that object. Image-based objects also have a GRID attribute that points to the grid that the objects were derived from. This attribute lets a process link the symbolic object to the label plane associated with other objects derived from the same image. In addition, all of the image-based objects have a POINTS attribute that contains all of the locations in the object. The spatially tagged parts of both objects are represented by a uniform location representation that lets us trade off between storage space, speed and fidelity while maintaining the ability to get all of the points in the object. When locations are represented by other objects, the locations of the objects are used as the location of the composite object. All of the representations can be converted to points and vice versa. This lets us write most functions in terms of points rather than depending on a specific location representation. Perceptual objects also have methods associated with them for performing geometric manipulations of their locations including rotation, scaling and translation for image-based matching operations. Associating the geometric operations with objects insures that all of the locations in the objects stay in registration.

```
(filter 13
    (near-object 'curve' 10)
    (within-filter :avg-intensity 3))
```

RETURN ALL CURVES IN GROUP 13 (AN APPLICATION SPECIFIC DATABASE) THAT ARE WITHIN 10 PIXELS OF THE CURVES POSITION IN ITS LABEL PLANE AND ARE WITHIN 3 OF THE OBJECTS AVERAGE INTENSITY

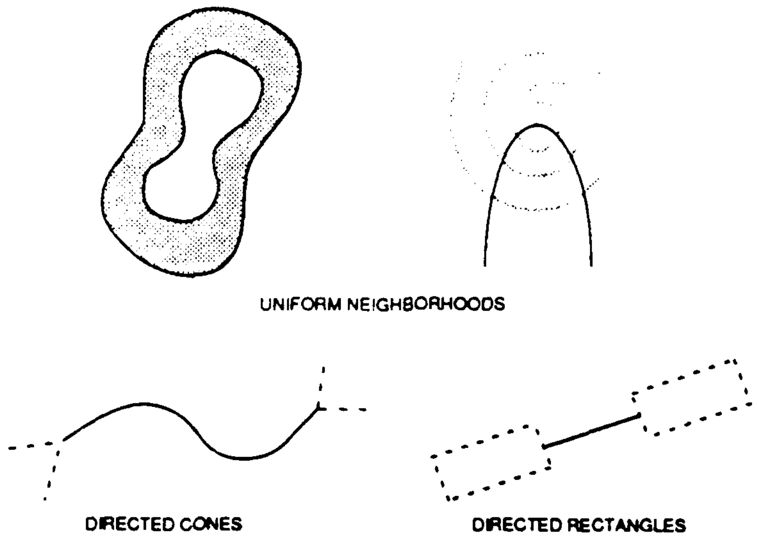


Figure 5-4: Spatial Query Language

5.1.1 Initialization of the PSDB

Currently, several default operations are performed to initialize the PSDB prior to grouping operations. These are primarily for contour related information for an image.

1. Red, green, and blue images are smoothed multiple times using the Burtian Pyramid reduction/expansion on each component.
2. The gradient is determined for each component image.
3. The Canny edge operator [Canny - 83] is applied to each intensity image at each spatial resolution.
4. The edges are thinned, traversed, and extracted as objects in the PSDB. A simple linear decomposition is associated with each curve and stored as in a tree structured attribute. The mean and variance of values along the curves are extracted with respect to intensity, contrast, and the gradient in each component and intensity. Orientation is associated with each linear subsegment of a curve.
5. Histograms are computed with respect to the curve object attributes. The image is divided into nested square subareas and histograms are associated with each of these.

6. Objects are sorted by attributes such as size, mean and variance of contrast and intensity, and extent of deviation from a linear approximation.

5.2 Grouping Processes

Grouping processes for an interface between queries produced during instantiation of an object model and image processing routines and sensors on board the telerobot. Examples are such things as:

- Find an edge parallel to a specified edge
- Find all connected green regions in an indicated area
- Match a predicted contour under specified curvature constraints
- Find a repeating pattern of edges
- Find a set of symmetric curves with contrast in a specified range

The specification of a grouping process corresponds to predicting a type of pattern of image features in an image and returning an ordered set of candidate patterns. The specification of a predicted group will be associated with the instantiation demons in an object model and instantiate the portion of the constraint network concerned with image feature to object model correspondences. A group itself is described by a set of parameters, the most critical of which will be set by the actions of the human when he places an object in the world model and it is projected against the image.

The measure-based grouper described here is a generalization of edge-linking procedures developed by multiple researchers [Martelli - 76]. It involves selecting a structure in the PSDB and a grouping criteria which can be expressed as a unified measure to evaluate sets of objects for satisfying the group-criteria. An example of such a grouping criteria is maintaining constant curvature and contrast along a set of ordered curves. The grouping process is expressed by several different types of functions for controlling the search and generating potential group members. These are functions for specifying an initial hypothesis, a successor function, an evaluation function, a goal function and a result function.

The initial hypothesis is usually a single primitive object, like a curve or region, although a group structure can also be used. This is selected by the best match to some prediction of image attributes derived from a model (such as finding a straight edge of high contrast at some expected orientation and position) or directly by the human establishing a correspondence between a PSDB object and an object model component.

Successors to a given hypothesized group are generated by applying a general constraint to all the potential objects. These global constraints range from attribute checks to complex geometric restrictions based on what can be globally inferred about the structure we are trying to find. The successor functions use a group, the possible successors, and a local constraint function that accounts for local changes in the group structure, to generate a list of new hypotheses. The local constraint functions have proven to be very powerful in that they allow us to deal with local changes rather than requiring a broad global constraint.

The evaluation function looks at a hypothesized group and evaluates it in terms of the criteria appropriate to the structures we are trying to find.

The goal function examines each hypothesis as it is pulled off the agenda and indicates whether in terms of the overall goal, the hypothesis should be opened or closed. The goal function also determines if the search should terminate and whether a hypothesis is itself a goal structure. The result function takes the results of the search and generates a permanent record.

Given the initial hypothesis, candidates, successor function, evaluation function and goal function, the search function performs a best first search over the hypothesis space by the following algorithm.

1. Pick the best hypothesis so far as ranked by the evaluation function.
2. Call the goal function to determine if this hypothesis is a goal and whether to continue with the hypothesis, terminate the search, or retrieve the next best hypothesis.
3. Call the successor function on this hypothesis to generate a new hypothesis.
4. Call the evaluation function on each of the hypotheses and then add them to the agenda ordered by their evaluation.
5. Repeat from 1.
6. When the search is ended, call the result function with each goal hypothesis and collect the returned values.

5.2.1 Running the Grouper

The grouper can run interactively in an automatic bottom-up fashion directly under human control, or top-down based upon the predictions associated with instantiated objects. To run the grouper, the following function is called:

```
(find-groups <group type> <label plane>
&key (filter) (interactive) (display)
(show 0) (redisplay t) (zoom '(1 1)))
```

The <group type> is the class of group which is being searched for. The <label plane> is the data structure which contains the results of perceptual processing which will be used as possible components of the desired groups. A filter over objects in the label plane can be specified at run time to limit the search as desired. By default, the results of grouping are deposited in the label plane for future processing.

5.2.2 Defining a Group

A group can be represented as a collection of hierarchies (Figure 5-5). Specification of a new group or an instance of some defined group involves the selection of elements from these hierarchies. Figure 5-5 shows the top level elements of the group which should be specified.

5.2.2.1 Component Constraints

Groups are made of other groups and perceptual objects in PSDB. There may be multiple types of components for a single group. For example, a group might contain both curves and junctions. The *type* slot indicates the component object type and is chosen from the group hierarchy (Figure 5-6). When attempting to group objects in the label plane, MBG only considers objects of the appropriate type.

5.2.2.2 Global Constraints

For each component, one can specify *global constraints* which must be satisfied by all potential components of a given type. For example, for curves to be considered as possible components of a straight line group they may be constrained to be of a certain length or straightness, or have attributes within some maximum deviation from other members of the group. These constraints usually take the form of filter functions which return a reduced number of candidate objects.

5.2.2.3 Successor Constraints

Given a partial group definition and some potential candidates in the label plane, MBG must select a subset which will be considered for addition to the group. The *successor constraints* specify how to choose new components to be added to the current group. For example, a successor constraint might be used to find all edges in a specified geometric relation to a hypothesized group. Most of these filter functions

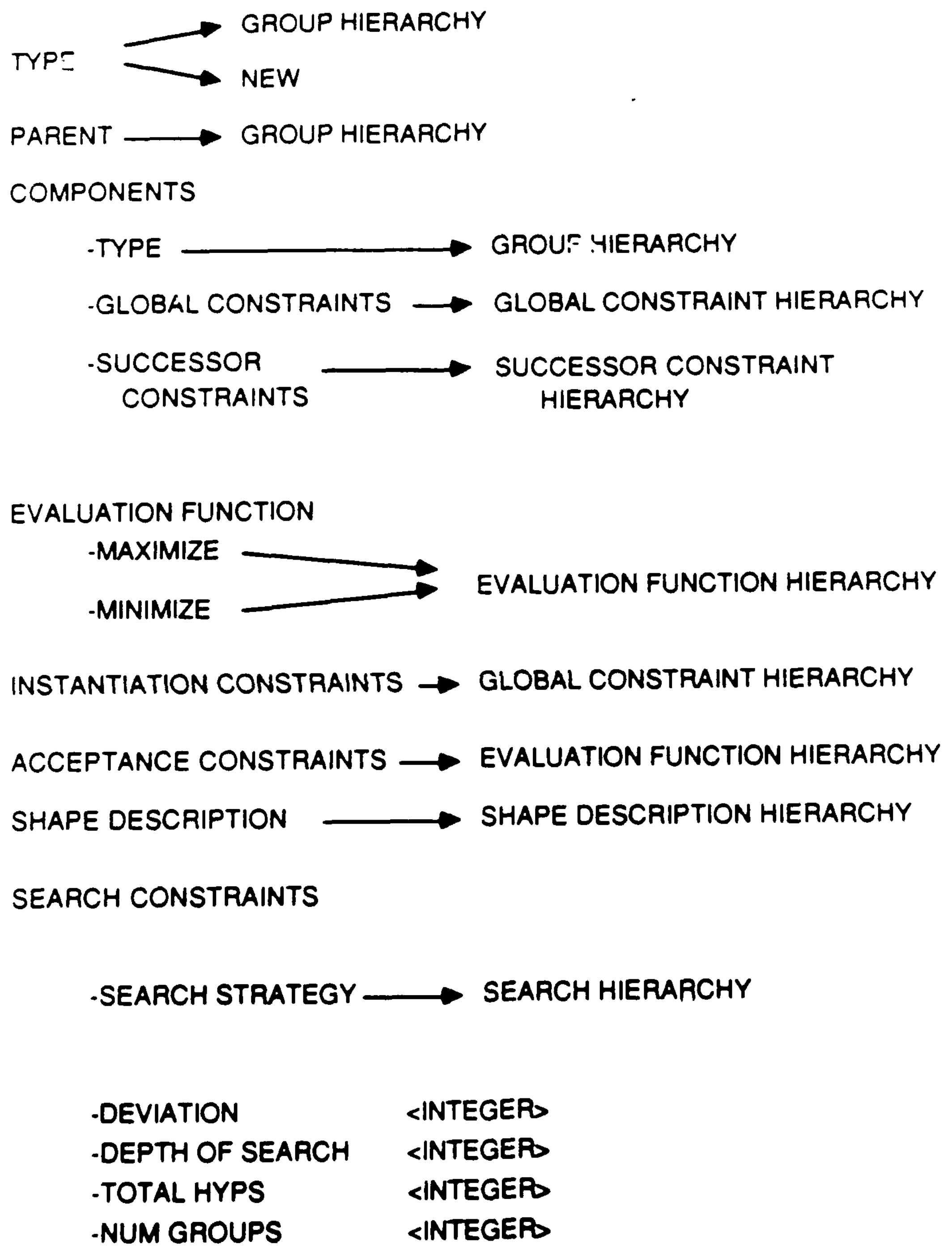

TYPE → GROUP HIERARCHY
TYPE → NEW
PARENT → GROUP HIERARCHY
COMPONENTS
-TYPE → GROUP HIERARCHY
-GLOBAL CONSTRAINTS → GLOBAL CONSTRAINT HIERARCHY
-SUCCESSOR CONSTRAINTS → SUCCESSOR CONSTRAINT HIERARCHY
EVALUATION FUNCTION
-MAXIMIZE → EVALUATION FUNCTION HIERARCHY
-MINIMIZE → EVALUATION FUNCTION HIERARCHY
INSTANTIATION CONSTRAINTS → GLOBAL CONSTRAINT HIERARCHY
ACCEPTANCE CONSTRAINTS → EVALUATION FUNCTION HIERARCHY
SHAPE DESCRIPTION → SHAPE DESCRIPTION HIERARCHY
SEARCH CONSTRAINTS
-SEARCH STRATEGY → SEARCH HIERARCHY
-DEVIATION <INTEGER>
-DEPTH OF SEARCH <INTEGER>
-TOTAL HYPS <INTEGER>
-NUM GROUPS <INTEGER>

Figure 5-5: Group Specification Hierarchy

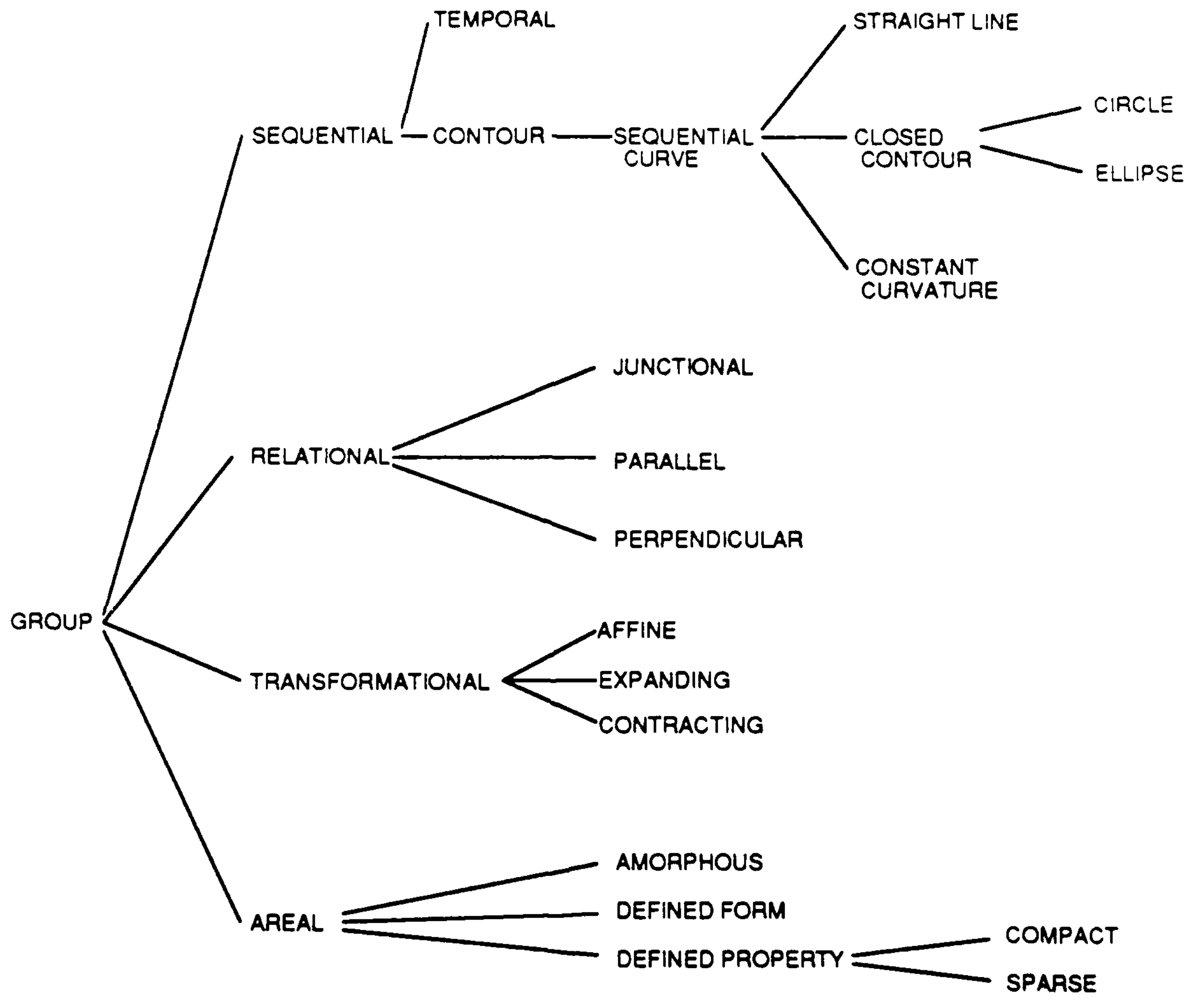


Figure 5-6: IS-A Hierarchy of Groups

work directly on the object representations, but some of them make use of the label plane for more complex geometric operations.

5.2.2.4 Evaluation Functions

Groups can evaluate themselves and obtain a value that indicates how "good" they are. This is used by search processes to guide the completion of groups. The group value is a linear combination of functions applied to the group components. These functions relate to the desired properties of an ideal group member in a given class. There may be attributes which should be *maximized* and some which should be *minimized* to achieve the best group. For example, for a straight line group, the idea is to maximize the length of the group while minimizing the average deviation from the best straight line through the curves.

5.2.2.5 Search

Various search strategies are available depending on the type of grouping desired. Users can define new strategies as appropriate. The current strategies are *best-first*, *monotonic-search*, and *accept-all-candidates.*

Best-First:

Best-first performs a standard breadth first search of potential groups. In this case, "best" means the highest valued group as determined by the specified evaluation functions. At a given point, the highest valued group is chosen and expanded by applying the successor constraints. Hypothesis groups are created by extending the group to include each of the candidate successors. These groups are evaluated and added to the list of active hypothesized groups. A goal function determines whether or not to accept or reject a hypothesized group. If no goal is specified (the default) then the hypothesized group is accepted when no more potential successors to the group exist. This is generally the most useful of the defined search strategies.

Due to the combinatorial nature of this search process, it is necessary to provide ways of restricting it. This search can be controlled by the following parameters:

- Within-n-of-Best:
  - the amount that hypothesized groups are allowed to deviate from the best hypothesized group found so far. A small value here restricts the search space and speeds the search. It also may prematurely prune groups which may turn out to be very good.
- Depth-Of-Search:

  - This restricts the total depth of the search tree.

- Total-Hyps:
  - This sets a maximum number of hypotheses to be maintained at a given time. Only the best groups are kept for future expansion.
- Best-n-Groups:
  - This specifies the number of groups to be returned from the search process.
- Threshold:
  - This specifies a minimum value that a group must have to be considered for further processing.
- Acceptance Constraints:
  - The evaluation functions described above can be used to specify minimum acceptance criteria for hypothesized groups.

Monotonic-Search:

This strategy is a special case of best-first search. By setting "TotalHyps" to 1 and "Within-n-of-Best" to 0, best-first search will behave monotonically (i.e. only groups of monotonically non-decreasing value will be considered). This can be useful when the criteria for group membership are well specified. The resulting search does not suffer combinatorial explosion but for most cases is too restrictive. Notice that by using the best-first strategy and changing the "deviation" value to some negative value, one can force the possible hypotheses to be monotonically increasing in value.

5.2.2.6 Shape Descriptions

Groups can be described in terms of their components, but since they often correspond to perceptual events they may have some real geometric structure. Hence it is often useful to represent this structure explicitly and treat the group as a single object deposited in the label plane.

5.2.3 Example

Figure 5-7 is an image of sumac leaves and berries. The textural regions corresponding to the berries in the sumac image are described by the "sumac-berry" group generated by the visually salient features determined from a model of a sumac-berry (Figure 5-8). The texture regions were generated by selecting primitive regions with

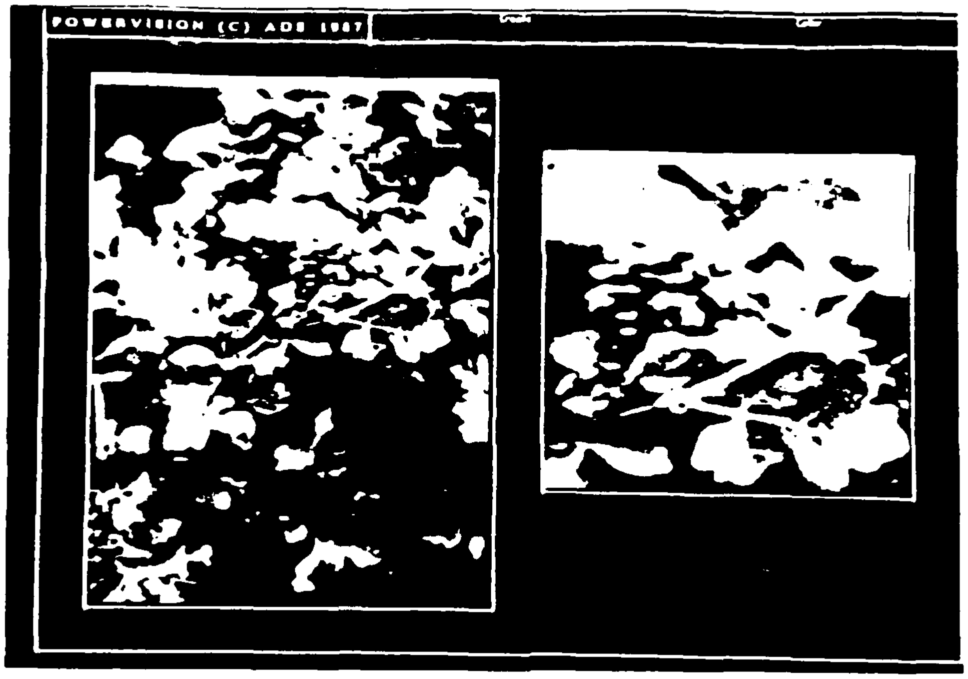


Figure 5-7: Sumac Intensity Image, and Berry Detail

attributes corresponding to sumac-berry substantial properties (Figure 5-9) and then applying region-based grouping processes to the adjacent regions. Textural regions which are compact and of a reddish color indicate possible regions of berries. Global component constraints limit the search for a berry group to regions of the appropriate color and contrast. Sumac berries tend to be in tight clumps hence new regions are added to the group only if they satisfy a compactness constraint. This constraint is specified as a maximum distance constraint on the centroids of regions in the group. Figure 5-10 illustrates the extracted regions and the berry group.

Figure 5-11 illustrates some straight line groups found in a ponderosa pine branch.

The straight lines of the pine branch image radiate from roughly the same location (the pine cone). This is captured by a "radial-line group" which specifies a roughly constant change in orientation among the elements of the group. Figure 5-12 shows the radial structure of the pine needles in red.

5.3 Perceptual Processing for Direct Extraction of Environmental Information

Many of the sensors on board the telerobot can be used to obtain environmental information directly without requiring significant segmentation or perceptual process-

```
(defflavor sumac-berries
        ((Type :sumac-berries)
        (components
           '((type 'region
                   global-constraints
                   '((:attrib-filter (and (attrib :avg-red 60 133) ;80
                                          (attrib :avg-green 15 101) ;30
                                          (attrib :avg-blue 22 113) ;35
                                          (attrib :var-red-contrast 25 1000)
                                          (attrib :var-green-contrast 26 1100)
                                          (attrib :var-blue-contrast 35 1155)
                                          (attrib :avg-red-contrast 12 48)
                                          (attrib :avg-green-contrast 11 50)
                                          (attrib :avg-blue-contrast 13 51)
                                          (attrib :area 0 93))))
                   successor-constraints
                   '((:centroid-distance 0.0 10.0)))))
         (Maximize '((:num-components 1.0)
                     ((:sum-component-attrib 'region ':area) 1.0)))
         (SearchStrategy ':best-first)
         (within-n-of-best 0.0)
         (totalhyps 1)
         (InstantiationConstraints
           '((:attrib-filter (type-filter 'region)))))
        (group-schema))
```

Figure 5-8: Sumac-Berry Region Group

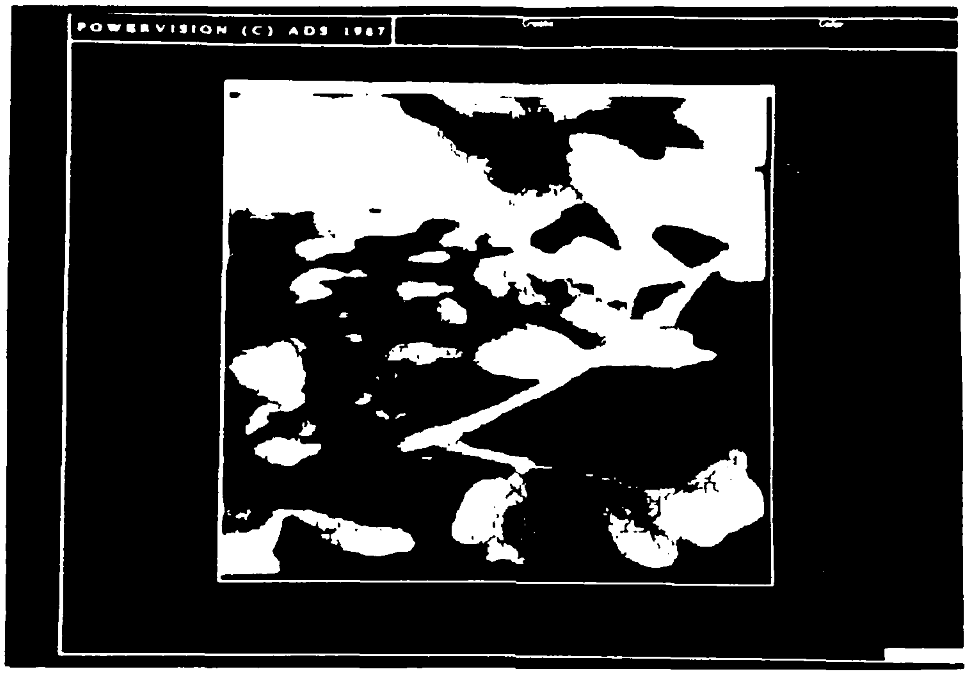


Figure 5-9: Primitive Regions in Berry Detail

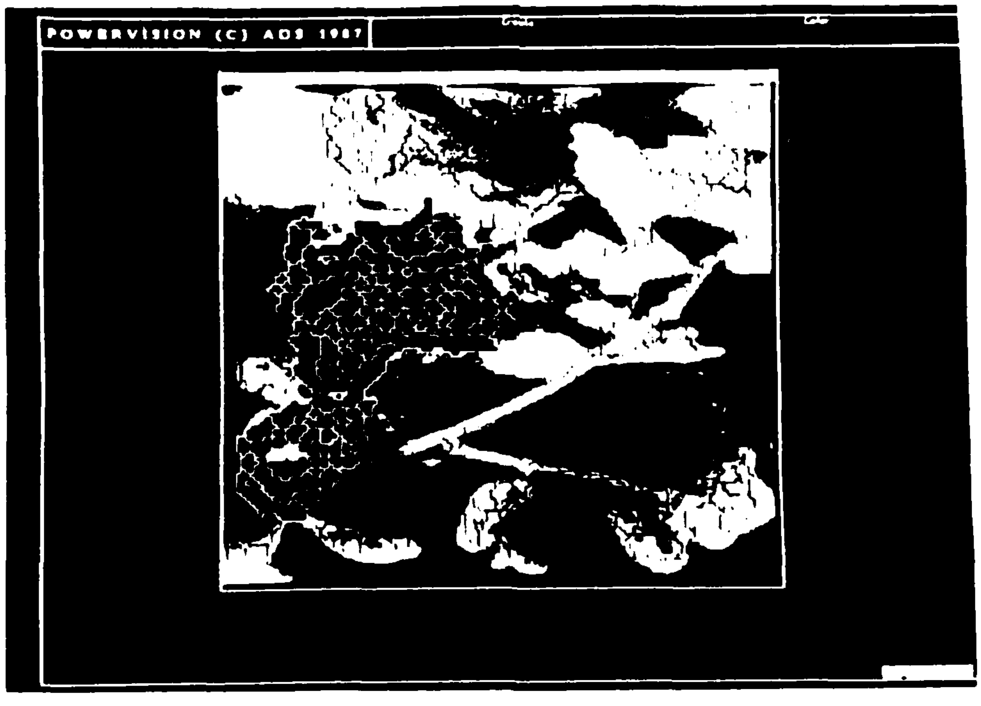


Figure 5-10: Regions Forming Textural Area

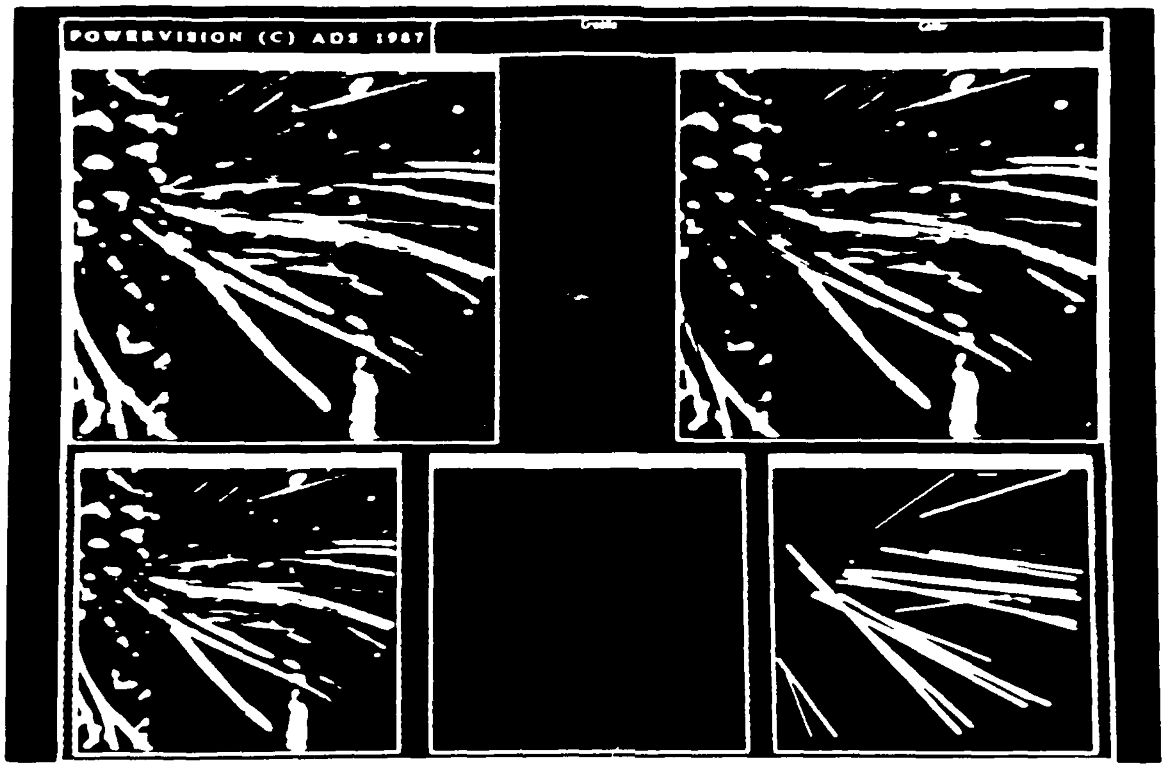


Figure 5-11: Straight-Line Groups in Ponderosa Pine Branch

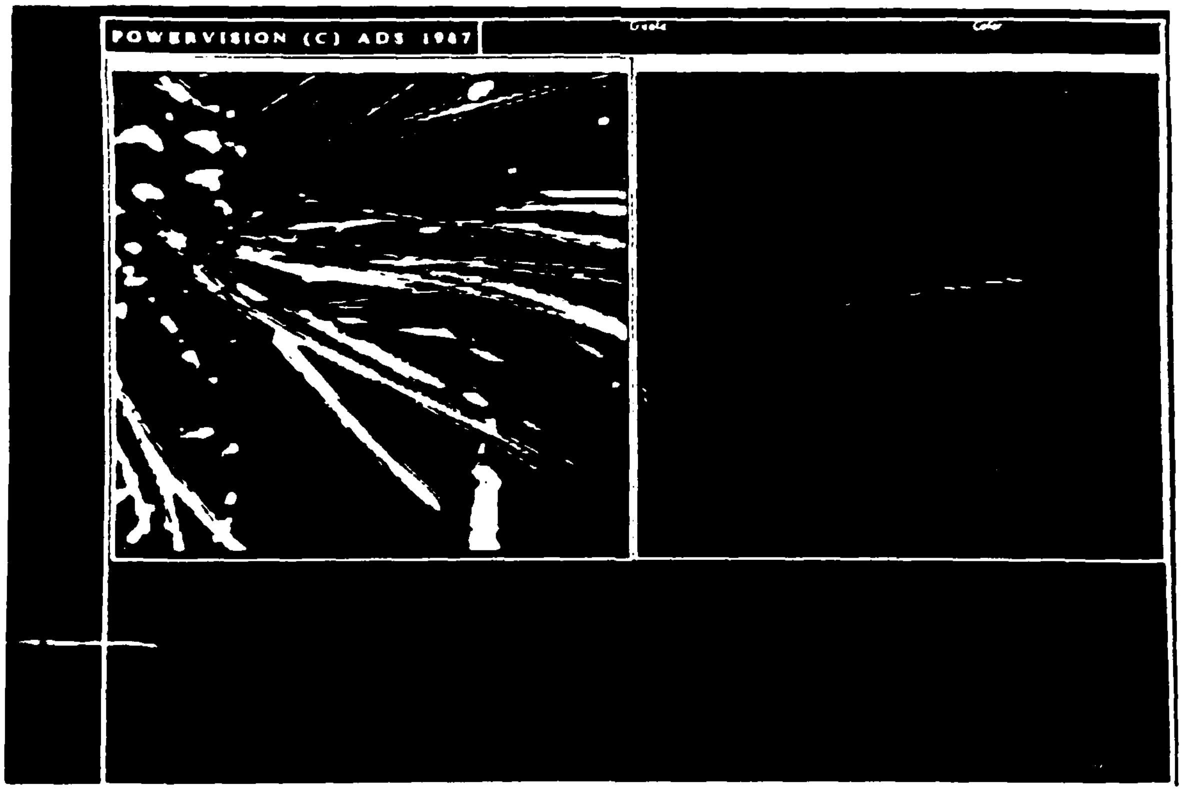


Figure 5-12: Radial Line Group

ing. This involves information about depth and surface orientation. This information can be deposited directly into the constraint networks for intrinsic object attributes and sensor feature correspondence. It is useful to think of such perceptual processing organized as a library of subroutines used for the direct extraction of environmental information which can be invoked by the constraint controller. Examples of such routines are:

- Scanning along a given direction and getting a depth profile
- Scanning across an edge to find depth discontinuities
- Scanning over an area to determine depth and variance
- Scanning along a ray of projection in three space

A key motivation for such routines is that they can be developed to yield very reliable depth information about a focused part of the environment, instead of forming, for example, a complete depth map of a scene which may have significant errors. The use of such routines will decrease computational expense. It also allows the instantiation of object models to direct the extraction of important information and avoid non-essential information. Such routines could be realized directly with the directable range sensor described in Section 2. It should also be possible to form a library of such routines based upon passive techniques, notable multi-camera stereo.

Passive techniques are essential for avoiding detection and for operation in unconstrained, outdoor environments. They also have benefits for eventual realization in parallel hardware architectures. In future project work, the algorithms for using several inter-related passive sensing techniques for the extraction of three dimensional scene information and performing model matching will be developed. This will include (but is not limited to) the use of inverse perspective, camera calibration, stereo, and model based stereo.

Multi-camera calibration enables measurements made from one camera to be related to those made with other cameras. It is used to determine the transformations between multiple sensor coordinate systems, the world model, and instantiated object models. Camera calibration is essential to solving the inverse perspective and stereo inferences needed in the following tasks. It enables the rays of projection from one camera to be projected onto the view of another camera to yield epipolar constraints necessary for processing multiple images.

There is a well established body of techniques for camera calibration [Castleman - 79]. In addition, there has been much recent work by which a robot can simply and robustly calibrate it's own sensors by simple standard moments [Brooks, Flynn and Marill - 88], [Tsai and Lenz - 88a]. This is essential for the semi-independent functioning of a telerobot which is separated by potentially large distances from a human controller.

Passive inference techniques can also be based upon the various constraints and relationships used to describe object models as developed under perspective projection with multiple cameras. This involves taking a geometric relation and expressing it using an inverse perspective transform [Lawton - 80] (see Section 4) consistent with multiple calibrated cameras. This yields a set of equations expressed totally in parameters that can be measured from images and whose solution determines consistent environmental depths for the associated image points. In general, for a single sensor and an arbitrary geometrical relation, this does not produce a solvable, or sufficiently constrained, set of equations. But for restricted geometrical relations and the use of the epipolar constraint between multiple calibrated cameras, an overconstrained set of equations is generally produced. Examples of such solvable relationships include relationships such as parallelism between lines; line segments with known lengths; line segments with known relative directions; points constrained to lie in a given plane; known angles between line segments and planes; and several others. Thus, for models described in terms of such relationships, the inference of depth under interactive registration with image locations is solvable. For these restricted cases, a prior knowledge of shape or the underlying geometry is theoretically sufficient to instantiate an object model.

Focused depth inference can also be obtained using stereo and motion. These techniques will be based upon triangulation using multiple cameras and can be the basis of a passive, directable range finder. The particular cases are:

- A human selects a point in one image and interactively determines its position in images obtained from other cameras under the constraints provided by camera calibration.
- A human selects a point in one image and it's position in other images is determined automatically under the constraints provided by camera calibration. The matching will be done using a small image area around the selected point which is treated as a feature. These features will be displaced and correlated along epipolar lines. The type of matching is similar to that used in [Lawton, Rieger, and Steenstrup - 87] for determining image displacements along translational flowpaths during sensor motion.
- A human selects a point in one image and the associated camera is able to move to another position. While it does, it will continuously track the selected environmental point and perform triangulation. This will simplify the matching processing and allow for optimizing the length of the base-line used for triangulation.

6. Long Term Database

The long term database (LTDB) is a database and a set of access functions that store and recall the following information:

- a priori knowledge about the world including maps, geographic grid databases, terrain, etc.
- sets of visual or other sensor observations, the inferences drawn from these observations via image processing, and previous human directed scene interpretations.
- the spatial and semantic relationships between observations, and inferences.

The basic function of the LTDB for the telerobotic vision system is to generate predictions of objects and terrain in a scene using a prior terrain information and previous scene interpretations. These predictions will be used to automatically suggest critical landmarks for a human to match and refine. This will enable the robot to more rapidly interpret a scene that it has visited previously and further reduce the processing load on the human.

Most of our work for implementing the LTDB was developed as part of the theory of **Qualitative Navigation** [Levitt and Lawton - to appear]. The first part of this theory was based on a multi-level spatial representation for an exploring robot (or in our case, telerobot) (Figure 6-1). The levels are ordered by the abstractness of represented spatial information. At the lowest (and least abstract) level in Figure 6-1 is a grid based representation of terrain described relative to a single fixed, global coordinate system. This corresponds to the type of information associated with an a prior terrain grid. The middle-level consists of locally coupled polar coordinate systems called **viewframes**. Each viewframe stores the direction and range of extracted landmarks and objects from a coordinate system based on the robot. There are intervals of uncertainty associated with the range and angle of the perceived landmarks in the viewframe. The use of the viewframe data structure allows us to conditionally attach it and its associated landmarks to a region of the global coordinate grid to reflect uncertainty in global position. The most abstract level of representation consists of viewframes with out any range information. This level corresponds to a purely topological representation of space.

The use of these multiple levels of representation allows us to cleanly combine two different types of information. Generally, the a prior terrain map is an array of numbers described relative to a fixed global coordinate system while the world and landmarks surrounding the robot are best described as distinct objects which are not easily summarized as a feature value which describes a 30x30 meter patch of earth. Both types of information are represented and related in the LTDB.

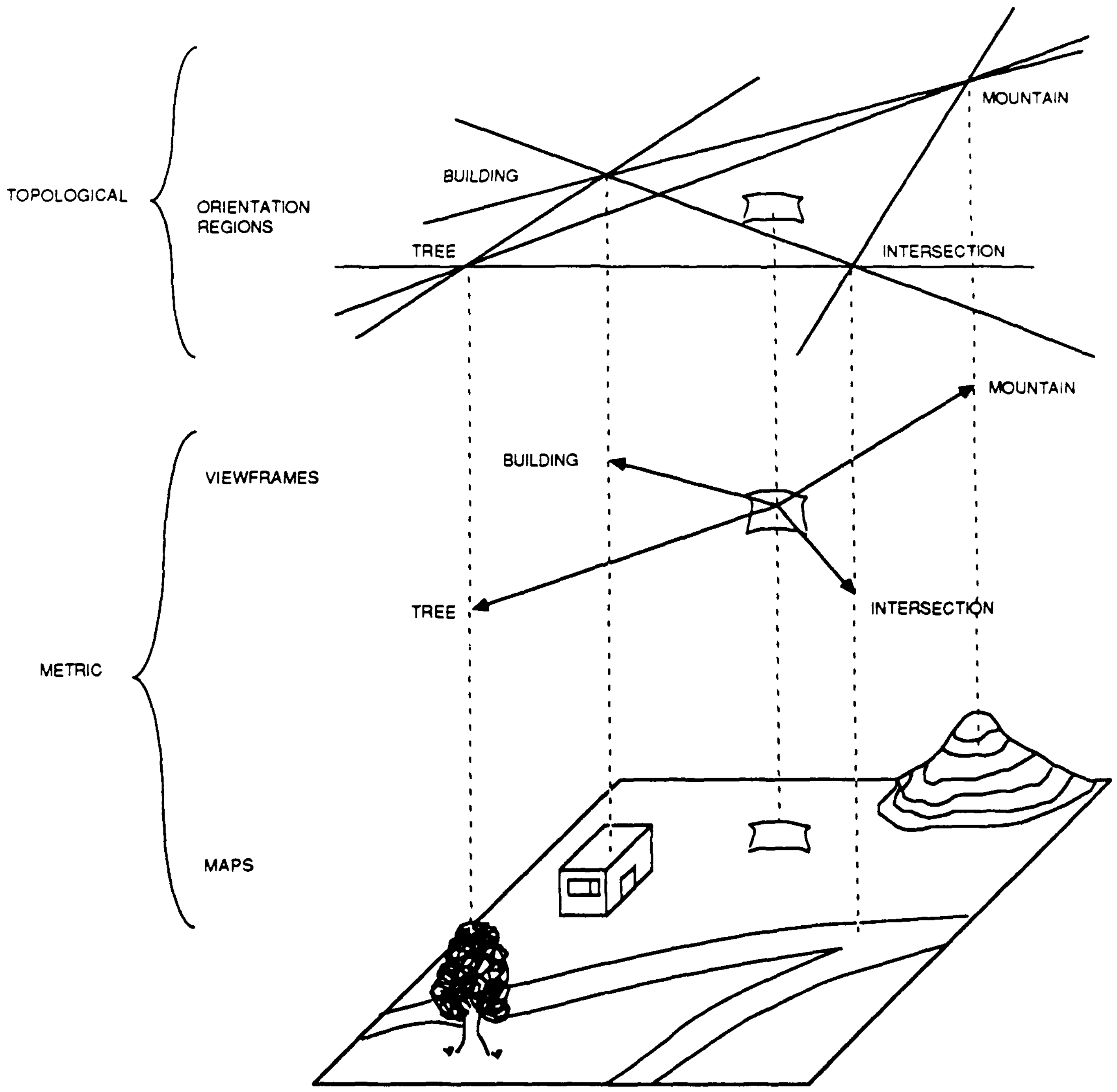


Figure 6-1: Multiple Levels of Spatial Representation

The second part of Qualitative Navigation was a technique of localization by which it was possible to determine the position of a viewframe from another viewframe with which it shares landmarks. What was significant was that this localization procedure was adequate for path planning and navigation even with enormous amounts of uncertainty (up 1000 percent) in landmark portion.

For the telerobot, this is used in several ways. Generally, the robot will have very good information about where it is if it can use the Global Positioning System (GPS). Thus, visual predictions are generated from the a prior terrain map for automatic registration and refinement by the human. Second, if the robot is near a place where a viewframe was extracted during a previous interpretation, the landmarks from that viewframe are available for interactive registration by the human. Using the viewframe localization procedure, once a few landmarks are registered, the others can be positioned automatically. The viewframe localization procedure would also be of use if GPS were not available.

The architecture for LTDB is pictured in Figure 6-2. LTDB contains several types of information: a priori terrain grids, a priori cultural feature networks such as roads and rivers, viewframes derived from previous interpreted scenes, landmarks, and viewpaths.

Landmarks are the distinctive visual events either observed at runtime or available from memory. These are chiefly pointers to instantiated object models in previously extracted world models. Landmark data includes time of acquisition, viewframes it occurs in, pointers to map or grid data tying the landmark to absolute coordinate systems (if available), as well as perceptual data. Viewpaths are sequences of viewframes acquired while the robot was moving through its environment. They are the key structure of visual memory. Each viewframe points at the landmarks it records. Because the landmark and viewpath databases point at each other, all modules that use LTDB have access to both databases. Terrain grids and cultural feature networks constitute the a priori knowledge available to LTDB. They are used to make visual predictions on paths, and for post-run processing for algorithms that use LTDB as input for creating and updating maps of the environment the robot has passed through.

Viewpath maintenance is the process that attempts to match the re-acquisition of landmarks to infer closeness of visual places in memory. Thus, when we re-acquire landmarks, our current viewpath is linked to existing ones in memory. This is the primary function that makes LTDB useful for visual predictions and for path planning.

The spatial reasoning module computes relationships between objects in visual memory in response to tasks and queries from the vision system. Fundamental operations include planning headings between the current viewframe and ones predicted in memory, path planning, and inference of global relationships between local coordinate systems.

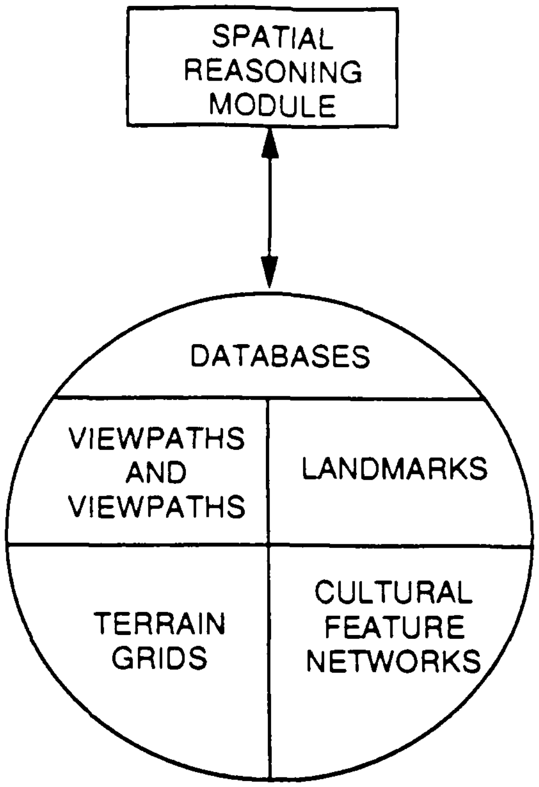


Figure 6-2: LTDB - Architecture

Following this introduction, Section 6.1 describes the prediction processing using an prior terrain grid. Section 6.2 describes the properties of viewframes.

6.1 A Priori Terrain Knowledge and Prediction

A priori terrain data is provided in several formats. Feature masks (or thematic overlays) are two-dimensional and follow standard Mercator projections onto the terrain. Examples include road networks, gullies, buildings, and fields. Digital elevation models (DEMs) are 2-1/2 dimensional in that they encode, at most, one elevation value at each point on the surface of the earth (so they cannot, for example, be used to depict tunnels).

Given a digital elevation model (DEM), we can generate various displays of data. These include contour diagrams, pseudocolor plots (where color is used to distinguish elevation or slope), or surface plots. It can be used as a synthetic image which acts as a prediction of what a sensor might capture when imaging the given terrain we have implemented. A perspective surface plot with hidden line elimination can be generated from arbitrary viewpoints using a camera model. The routine can plot arbitrary values into an image for subsequent processing or it can draw colors into a window for display purposes.

Given a DEM in the form of a grid, the surface plot routine takes two types of input parameters: positional and camera. The position of the "pseudocamera" is specified in terms of an x,y location (in grid coordinates), an orientation (in degrees where 0°is North or positive y in the grid), a height above the surface (in the same units as the values in the DEM), and a tilt angle (in degrees where 0 is straight ahead). This captures all the degrees of freedom (in an intuitive form) of a real camera in the real terrain. The "pseudocamera" is modeled by its resolution (in grid cells), field of view (in degrees), and range (the distance the camera can "sense" in grid units).

Given underlying surface features (such as roads, fields, buildings) that are registered to the DEM, a variation of the plot routines will write values derived from those objects onto an image. These values may be pointers pointing back to the descriptions of the objects themselves (making the image a label grid) or some derived property of the object (such as albedo). The resulting image is then available to be processed to form a prediction of what can be found in the scene.

The display routines can generate label plane images (see Section 2 and 5) where the values in the image are pointers to the geographic objects (e.g., roads, fields) that occur in the scene (Figure 6-3). Since the visual characteristics of these objects can be modeled, it is possible to use their position in the synthetic image as predictions as to what will be found in the sensor-derived imagery.

Given approximate positions of geographic entities in the image allows model-based processing to be carried out. In hypothesized boundary regions, edge detectors can be run that are tuned to the appropriate direction. Since there is a high probability that an edge is present, weaker evidence can be considered than would be the case with a model-independent detector. In regions corresponding to geographic entities, (roads, forests, etc.) texture and other region-based operators can look for evidence concerning what is known about the particular region. For example, if it is known that a region is a forest, then that indicates a texture with a high standard deviation. However, if the particular type of forest is known, and additional information exists (e.g., the time of year, sun angle, density of trees), then much more specific predictions about color can be made. If the region has been seen previously, then even more exact predictions can be made.

All the information regarding the synthetic image can be manipulated in the same way as sensor-derived images. The extracted regions will be placed in PSDB and have the same representation as other "regions". This facilitates matching the predictions with the image structures. The same type of queries over PSDB that have been previously described can also be used to manipulate this data. This consistency across different types of data simplifies the software engineering of the vision system.

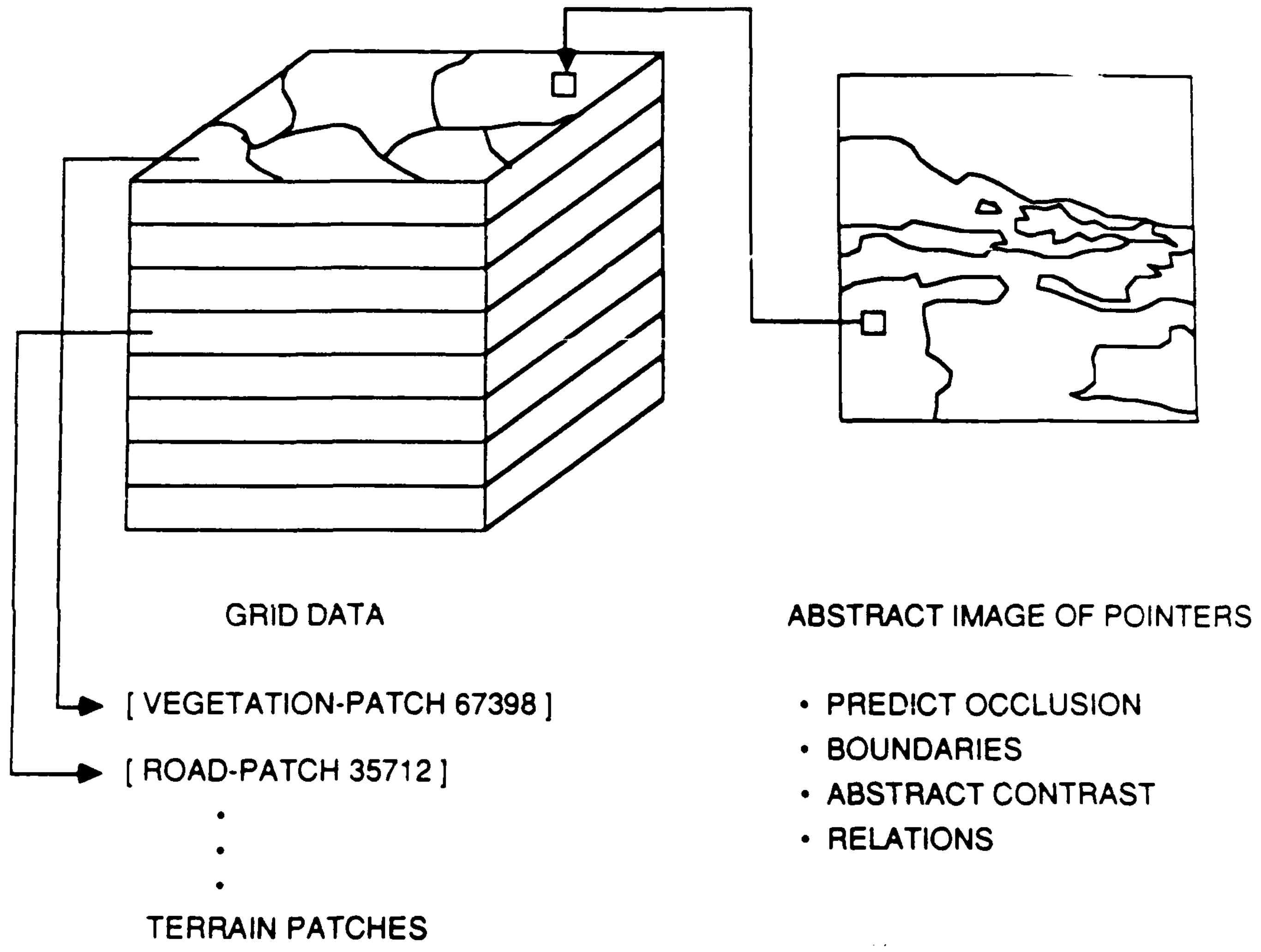
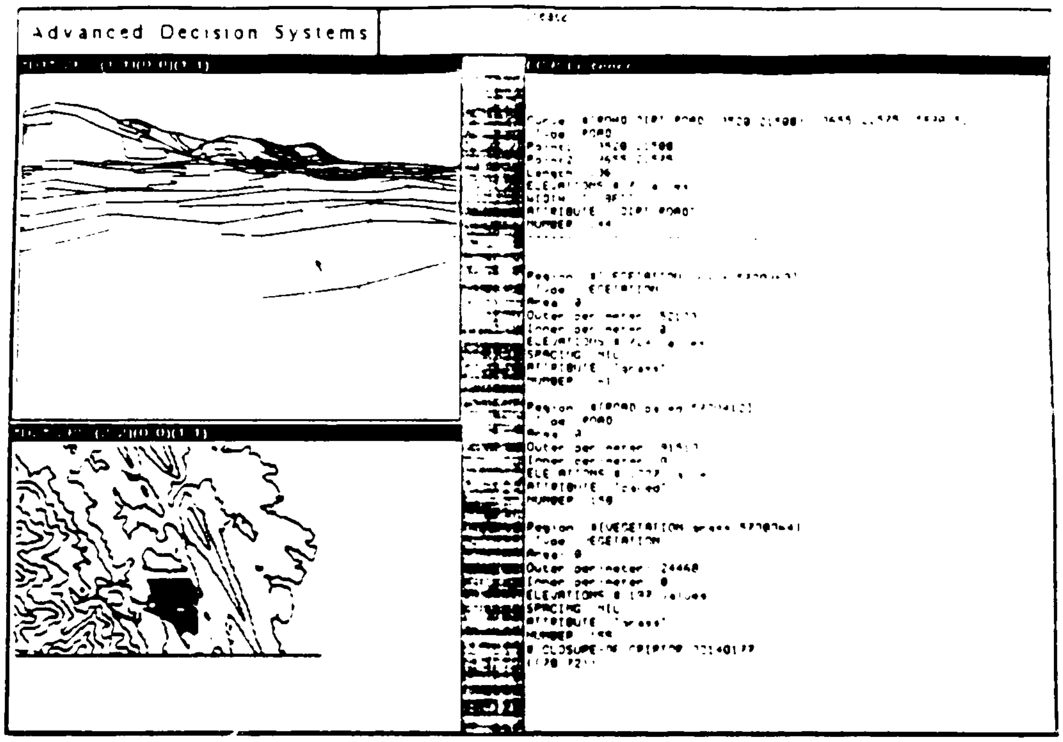


Figure 6-3: Label Plane Display of Terrain



```
Curve  #[ROAD [illegible] ROAD  [illegible]  [illegible]  [illegible]
 Type  ROAD
Point1   [illegible]
Point2   [illegible]
Length   [illegible]
ELEVATIONS # [illegible]
WIDTH  [illegible]
ATTRIBUTE  [illegible] ROAD
NUMBER  [illegible]

Region  #[VEGETATION [illegible]
 Type  VEGETATION
Area  0
Outer perimeter  [illegible]
Inner perimeter  0
ELEVATIONS # [illegible]
SPACING  NIL
ATTRIBUTE  [illegible]
NUMBER  [illegible]

Region  #[ROAD [illegible]
 Type  ROAD
Area  0
Outer perimeter  [illegible]
Inner perimeter  0
ELEVATIONS # [illegible]
ATTRIBUTE  [illegible]
NUMBER  [illegible]

Region  #[VEGETATION [illegible]
 Type  VEGETATION
Area  0
Outer perimeter  24460
Inner perimeter  0
ELEVATIONS # [illegible] values
SPACING  NIL
ATTRIBUTE  [illegible]
NUMBER  [illegible]
# CLOSURE-[illegible] [illegible]
[illegible]
```

Figure 6-4: View to Grid Attributes

Figure 6-4 shows human interaction with such a label plane image using a mouse. The window in the upper left corner of each of these figures shows the predicted image. The pointer in this image indicates a region that is being queried for the relevant types of objects projecting onto this area of the image. The figure on the lower left indicates the regions of visibility with respect to the position of the vehicle superimposed on top of a contour map display.

6.2 Local Coordinate System: Viewframes

We define a geographic "place" in terms of data about visible landmarks. A place, as a point on the surface of the ground, is defined by the landmarks and spatial relationships between landmarks that can be observed from a fixed point in space. More generally we can define a place as a region in space, in which a fixed set of landmarks can be observed from anywhere in the region, and relationships between them do not change in some appropriate qualitative sense. Data about places is stored in structures called **viewframes**.

Viewframes provide a definition of place in terms of relative angles and angular error between landmarks, and coarse estimates of the absolute range of the landmarks from our point of observation. Virtual axes between pairs of selected landmarks form a local coordinate system that can be used to localize the robot relative to all visible landmarks. These coordinate systems are truly local in that it is unnecessary to know anything about the absolute relationships between observed landmarks, or to relate the robot's current location to any past location the robot has passed through.

A viewframe encodes the observable landmark information in a stationary panorama. That is, we assume that the sensor platform is stationary long enough for the sensor to pan up to 360°, to tilt up to 90°, and to recognize landmarks in its field of view. Viewframes allow us to localize our position in space relative to observable landmarks in several ways. In performing a viewframe localization, we can make use of observed or inferred data about our approximate range to landmarks. Errors in ranging and relative angular separation between landmarks are smoothly accounted for. A priori map data can also be incorporated.

To generate a viewframe, relative solid angles between distinguished points on landmarks are computed using a sensor-centered coordinate system. A distinguished point on a landmark may be a statistically derived point such as the centroid of the projection of the landmark in an image, a structurally derived point such as a vertex or high curvature boundary point, or a perceptually derived point as in a unique visual feature of the landmark.

We establish a sensor-centered spherical coordinate system that fixes an orientation in azimuth and elevation and takes the direction opposite our current heading as the zero degree axis. Then two landmarks in front of us, relative to our heading, will have an azimuth separation of less than 180°. If we assume that no two distinguished landmark points have the same elevation coordinates (i.e., no two distinguished points appear one directly above the other) then we obtain a well-ordering of the landmarks in the azimuth direction which we can speak of as "ordered from left to right". The relative solid angle between two distinguished landmark points is now well defined. See Figure 6-5.

Under the above assumptions, we can pan from left to right, recognizing landmarks, L_i, and storing the solid angles between landmarks in order, denoting the angle between the i-th and j-th landmarks by Ang_{ij}. The basic viewframe data is this ordered list, $L_1, Ang12, L_2, Ang23, L_3, ...$ Note that the relative angular displacement between any two landmarks can be computed from this basic list. However, the angle between landmarks that is more readily measured is the planar angle, Θij, established by the sensor focal point and the two distinguished points observed in the i-th and j-th landmarks. There is an error in computing these relative angles that is at least as great as resolution of the vision system. The angular error is measured by e_{ij} between landmarks i and j. Finally, range estimates are required for landmarks recorded in viewframes. These estimates can be arbitrarily coarse, but finite. We only insist the true range lie between the bounds specified for the estimate. With this data in mind, the structure of a viewframe is given by $L_1, [r_1 1, r_1 2], Ang12, e_1 2, L_2...,$ where r_{11} is the nearest range estimate and r_{12} is the farthest.

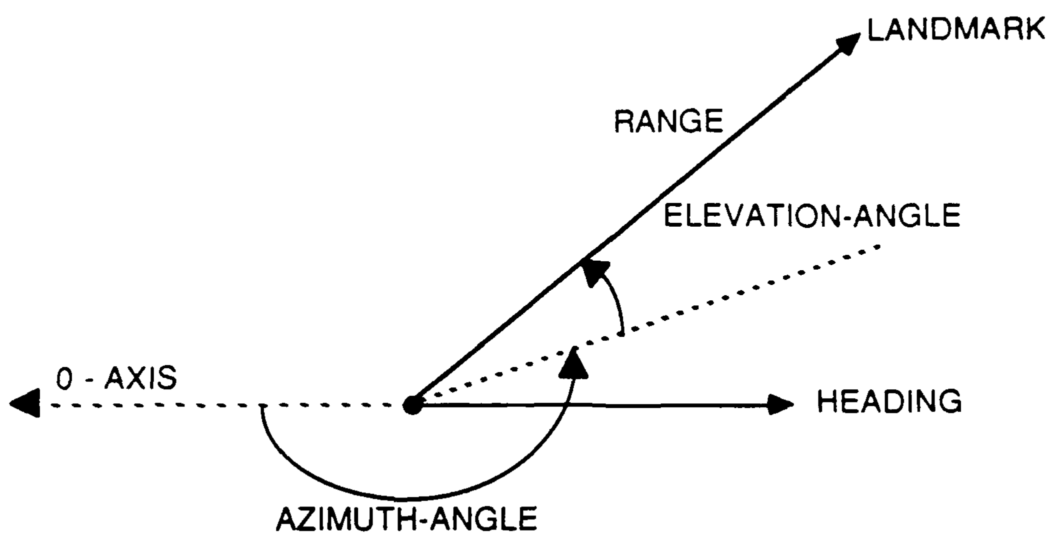


Figure 6-5: Sensor-Centered Coordinate System

6.2.1 Viewframe Localization

We now explain how it is possible to localize ourselves in space relative to these observed landmarks. This is used to determine and refine one's position given rough correspondences between predicted landmarks and observed image features. A more complete description can be found in [Levitt and Lawton - to appear].

We begin by noting that the set of points in 3-space from which we can observe an angle of Θ_{ij} between landmarks L_i and L_j is constrained to a torus-like space, pictured in Figure 6-6. This is more easily observed in a planar cross-section where the shape is the figure eight cross-section in Figure 6-7. To prove this, we set up a polar coordinate system with the origin at L_i, and with L_j at coordinates [s,0], where s is the fixed but unknown distance between the landmarks. Denote the (unknown) distance from the sensor to L_i by R_i and the distance to L_j by R_j. We now ask, what are the set of points in the plane, $[r, ang]$, from which we can observe an angle of Θ_{ij} between L_i and L_j? This situation is pictured in Figure 6-7.

We can now compute:

$$\begin{aligned} r_i &= s_{ij} \cos\phi + s_{ij} \cot\Theta \sin\phi \\ \phi &= \cot^{-1}\left[\frac{r_i}{r_j}\csc\Theta - \cot\Theta\right] \end{aligned}$$

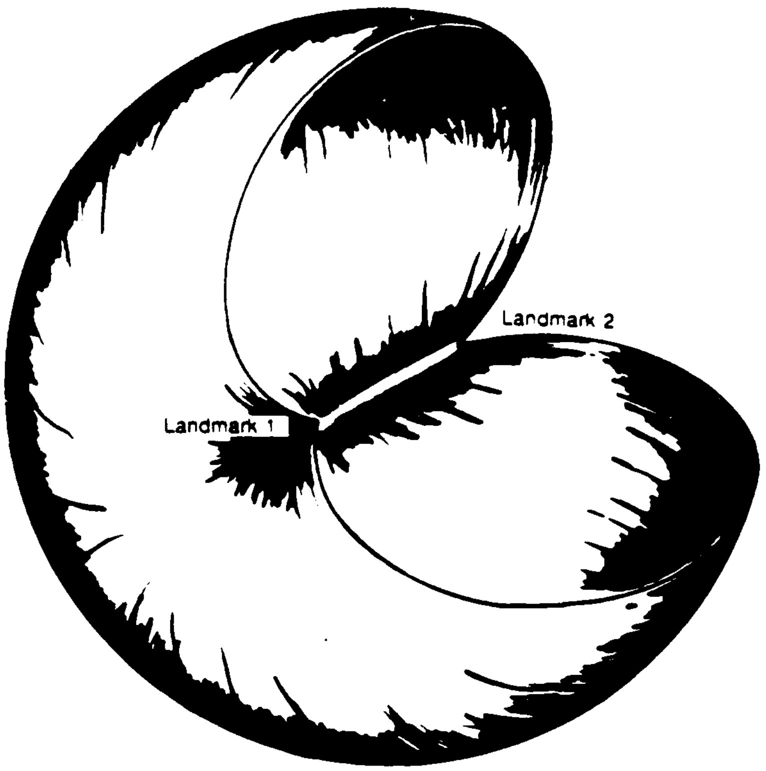
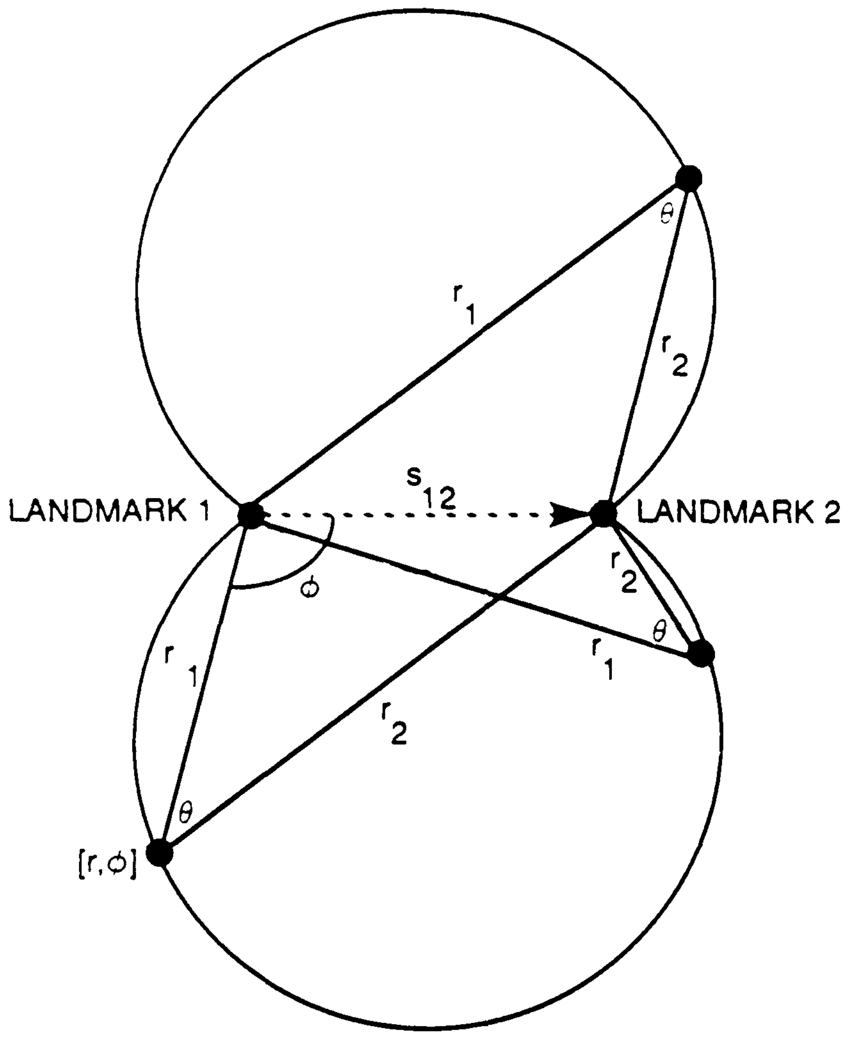


Figure 6-6: Constant Angle Toroid



$$\varphi = \cot^{-1}\left[\frac{r_1}{r_2}\csc\Theta - \cot\Theta\right]$$

$$s = \left[r_1^2 + r_2^2 - 2r_1 r_2\cos\Theta\right]^{\frac{1}{2}}$$

$$r = s\cos\phi + s\cot\Theta\sin\varphi\ (s\text{ , }\cot\theta\text{ constant})$$

Figure 6-7: Constant Angle Circular Arcs

This is the polar form for a circle with center $\left[\frac{s_{ij}}{2}\csc\Theta, \frac{\pi}{2} - \Theta\right]$ and radius $\frac{s_{ij}}{2}\csc\Theta$. It has singularities where R_i or R_j are equal to zero. By symmetry we obtain the figure-eight-like shape pictured in Figure 6-7. Rotating the circular arc in 3-space about the axis defined by the line segment joining L_i and L_j, we obtain the figure pictured in Figure 6-6.

Figure 6-7 shows how varying the sensor location along the circular arc is equivalent to varying the absolute ranges to the two landmarks. If we can bound the ranges to the landmarks, then we can localize ourselves along the circular arc accordingly. This is logically equivalent to establishing a local coordinate frame between the landmarks, and bounding our location relative to that frame. Note that if we can register the landmarks with a priori map data, then we can know and use the distance between the landmarks, but this is not necessary.

Error in angular measurement of the Θ_{ij} corresponds to different choices of concentric circular arcs each of which contains $[L_i, L_j]$ and the sensor focal point. If we union these arcs together, we obtain the localization, $Loci_j$, of the sensor relative to the two landmarks. Because our localization must be true relative to all observed landmark pairs simultaneously, the intersection of the $Loci_j$ over all i>j give our best localization. Intersecting the $Loci_j$ requires that the $Loci_j$ be represented in the same local coordinate frames. These concepts can clearly be extended to 3-space reasoning if necessary.

In summary, if we can observe distinguished points on landmarks and the relative angles between them, and if we can put bounds on the ranges to landmarks, then we can compute our location in space relative to pairs of landmarks, and intersect the pair-relative localizations to obtain our best localization relative to the entire set of observed landmarks.

6.2.2 Viewframe Headings

Headings and directions between viewframes can be computed between two viewframes that share at least two landmarks (two landmarks for 2D reasoning, three landmarks for 3D reasoning) in common. If the set of landmarks included in a viewframe destination are visible and we are very close to our original point of observation, then it is straightforward (up to traversability of the intervening terrain) to perform a hill-climbing algorithm to bring the sensor to the point of observation where the viewframe was previously collected.

An alternative approach is to formulate the viewframe localizations for each of the viewframes in the common local coordinate system defined by the two (or three) landmarks and the sensor. Note that the the two landmarks must not be co-linear with the sensor (the three must not be co-planar with the sensor). We then have two regions expressed in the same coordinate system. Any affine linear transformation that maps one viewframe approximately onto the other may be taken as a heading. For example, we can translate the centroid of the first viewframe to the centroid of the second. This is the definition of viewframe heading transformation we use as a default. It defines a heading as a vector pointing between the viewframes and supplies the vision system with an intuitive notion of "head thataway". A viewframe heading is not generally the same as a metric heading because the points in space that the sensor occupied when the viewframes were collected may not be mapped onto each other by the viewframe heading transformation. Figure 6-8 illustrates the generic situation in which we can compute a viewframe heading. One viewframe contains the landmarks L_1, L_2 and L_3, and the other viewframe contains landmarks L_2, L_3 and L_4. Range estimates to L_2 and L_3 may also be different in the two viewframes.

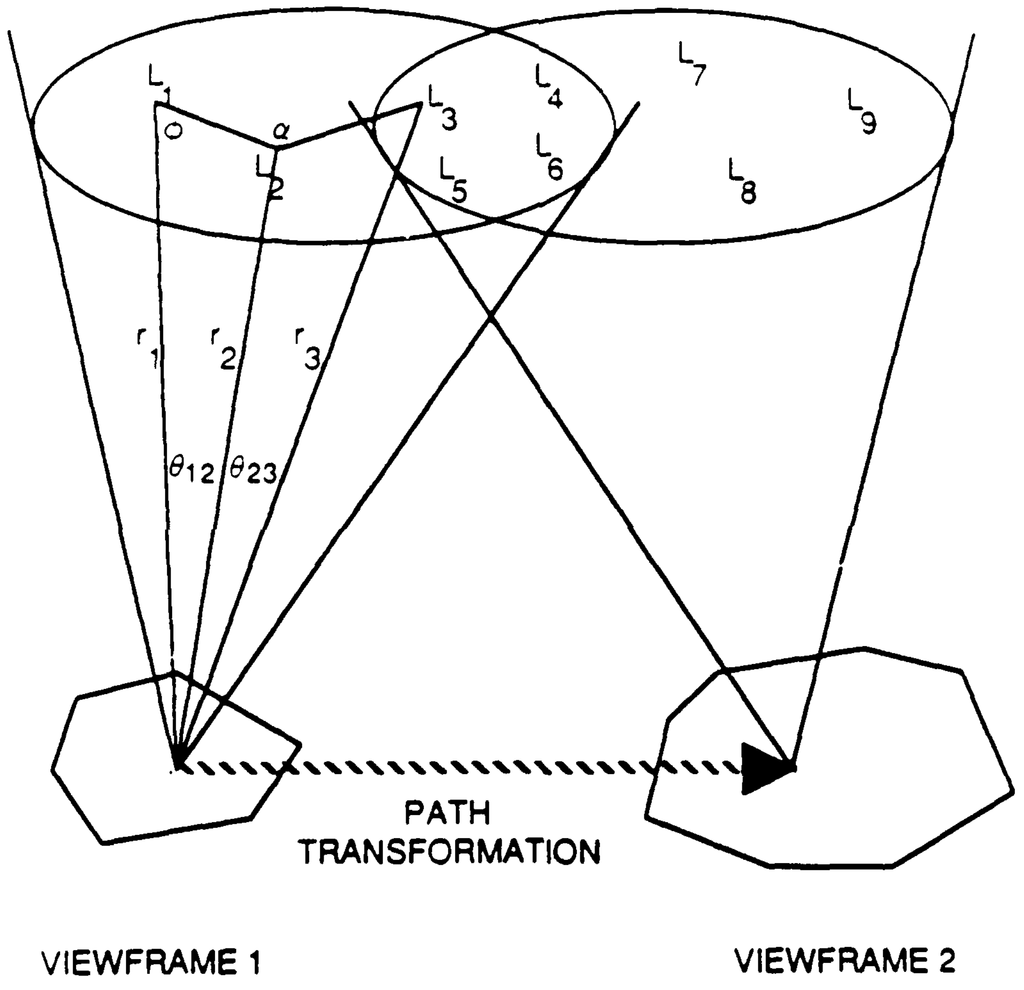


Figure 6-8: Viewframe Heading

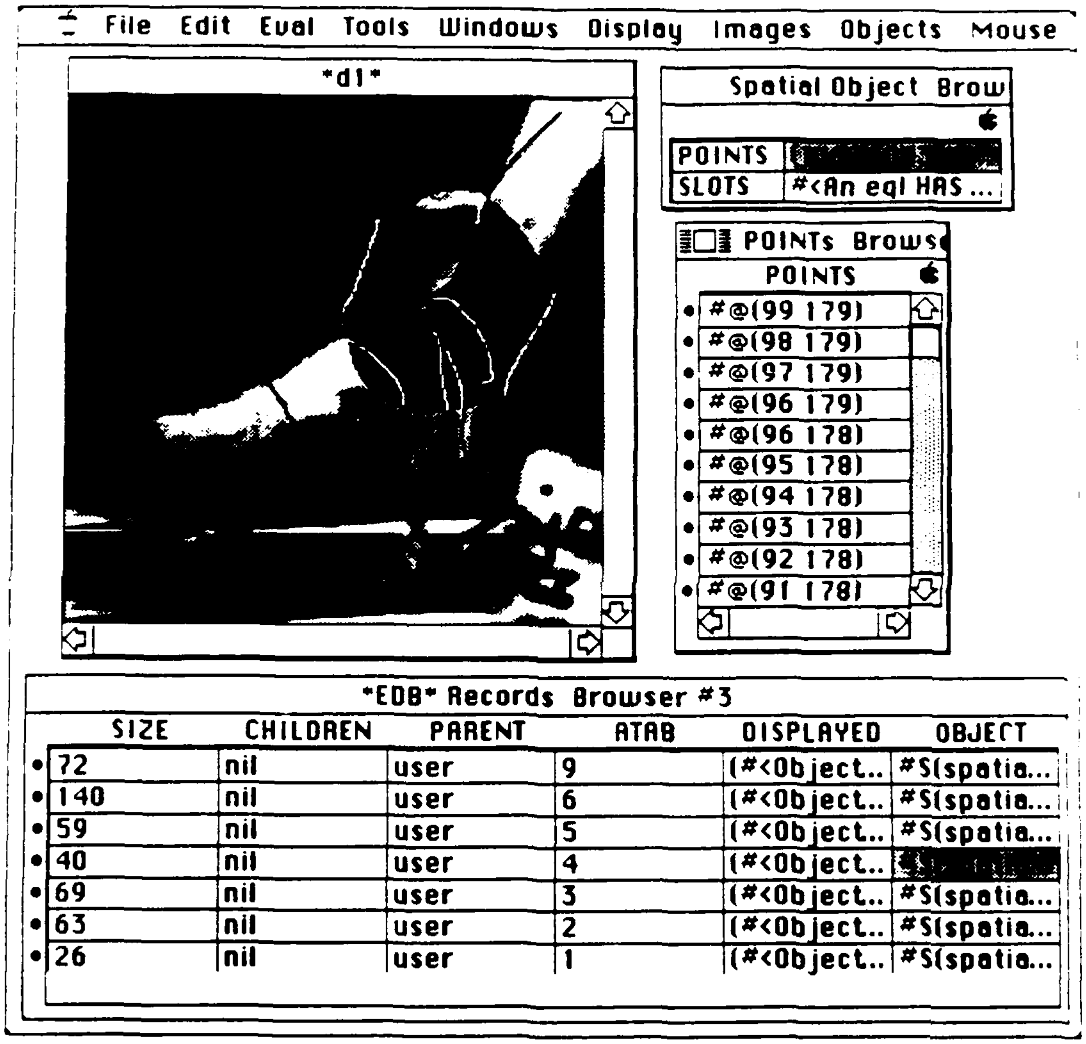
7. Mac II Based User Interface and Processing

We developed a prototype user interface for experimentation during Phase I work. It allows for basic actions such as viewing multiple images at different resolutions; manipulating three dimensional objects relative to displayed images while projecting the objects onto the images registered with the world model.

We view this initial interface as the first of a sequence. It is based on a few basic mechanisms such as display windows, color overlays, label-planes, and text browsers and utilizes standard input devices such as a keyboard, mouse or drawing pad. The user interacts with objects through textual browse tables which describe the values of the attributes of objects and present the explicit interconnections between objects. The interaction is a little painstaking because objects are manipulated by typing in values into slots in the text browsers and watching other values be propagated in displayed browse windows. This is a basic level of functionality which is useful for debugging. The next level of interface will involve interactive devices which enable several values to be set and operations to be performed simultaneously. This will include items such as data gloves, speech understanding hardware, and multiple displays. This will enable a user to specify many aspects of the world model simultaneously and also to use intuitive spatial gestures in developing the world model. An example would be having the physical sensation of grabbing a model and placing it into the world. This technology is currently becoming readily available and inexpensive. The final stage of the interface will be a virtual reality style presentation of the work model which the human actually inhabits the world model as it is established.

Our initial work was done on a Mac II using Coral CommonLISP [Bobrow et al. - 87]. Our choice was based upon several factors. There is extensive commercial software available for the Macintosh for graphics with a growing emphasis on video and image processing. The Mac toolbox [Apple Computer - 88] supplies an enormous facility for building interfaces, all of which was accessible through Coral CommonLISP. There are also exciting developments in interactive devices for gestural and voice input for the Macintosh.

The basic components and constructs used in our environments have been described in [Lawton and McConnell - 88], [McConnell et.al. - 87], [Edelson, et.al. - 88], and [Riley et.al. - 87]. In general, we found that developing corresponding components of the user interface on the Mac was fairly simple. For example, Figure 7-1 shows the interactive application of a spatial mask to a set of edges registered with an image. The selected edges are then displayed in a Browse Table for further inspection. We found that several of the commercially available products could be integrated with image handling. The use of generic CommonLISP made possible the use of programming constructs developed for CommonLISP-based IU environments.



POINTS	[illegible]
SLOTS	#<An eql HAS ...

POINTS
#@(99 179)
#@(98 179)
#@(97 179)
#@(96 179)
#@(96 178)
#@(95 178)
#@(94 178)
#@(93 178)
#@(92 178)
#@(91 178)

EDB Records Browser #3

SIZE	CHILDREN	PARENT	ATAB	DISPLAYED	OBJECT
72	nil	user	9	(#<Object..	#S(spatia...
140	nil	user	6	(#<Object..	#S(spatia...
59	nil	user	5	(#<Object..	#S(spatia...
40	nil	user	4	(#<Object..	[illegible]
69	nil	user	3	(#<Object..	#S(spatia...
63	nil	user	2	(#<Object..	#S(spatia...
26	nil	user	1	(#<Object..	#S(spatia...

Figure 7-1: Selection and Browsing of Curve Objects

On the negative side, we found that integrating other commercial application software such as CAD/CAM, databases, and expert system building tools was still rather difficult and awkward. In part, this is due to the lack of a conventional operating system which supports such things as virtual memory, multi-tasking, interprocess communication, and the lack of language interfaces for much of the commercial Mac software. Another limitation that has made us question the appropriateness of the Mac is the Allegro Coral CommonLISP environment itself. While providing great flexibility and power as a development environment, it needs to be greatly optimized as a run-time system before we can consider it for developing computationally intensive IU environment applications. The main hardware limitations we experienced stem from an overworked CPU. The central processor must not only perform all the computation and symbolic processing required for image understanding, but must also drive all the graphic display operations that are so fundamental to this type of interactive image and graphical object oriented application and orchestrate the access of all devices to the main memory. The CPU handles all file writes and memory management tasks byte by byte. This problem can be solved in several ways, and rumors from Apple indicate that they are actively developing hardware that addresses all these solutions (e.g., using an AMD 29000 RISC graphics coprocessor increasing bus bandwidth from 10mHz to 20mHz, and having direct memory access channels).

The limitations we have discovered are, in all likelihood, soon to be addressed by Apple so they can compete with low-cost scientific workstations from other vendors such as Sun, DEC, Apollo, and Next. With the increasing sophistication of user-interface tools associated with these machines, increased power, and a large third-party supply of hardware and software, they all provide very attractive platforms for IU environments.

In this section we begin by describing our hardware set up (Section 7.1) and the (components of) the Interface Software in Section 7.2. We conclude with several examples of the interface and system components in operation.

7.1 Hardware Configuration

The Mac IIx and Mac IIcx are full 32-bit, 68030-based machines using a 15.67mHz clock, a 68882 floating point coprocessor, SCSI and RS-422 ports, and a flexible NuBus expansion bus architecture (10mHz). The Mac IIcx has three NuBus expansion slots, and the Mac IIx provides six NuBus expansion slots. The NuBus provides an open architecture which has led to widespread commercial availability of powerful and inexpensive extended hardware capabilities which include coprocessors, memory, storage, digitizers, extended ports, robotic control, measurement, voice recognition, and a rapidly growing assortment of other equipment. Because of the large commercial Mac market, these products are relatively inexpensive, powerful, simple to install, and well supported.

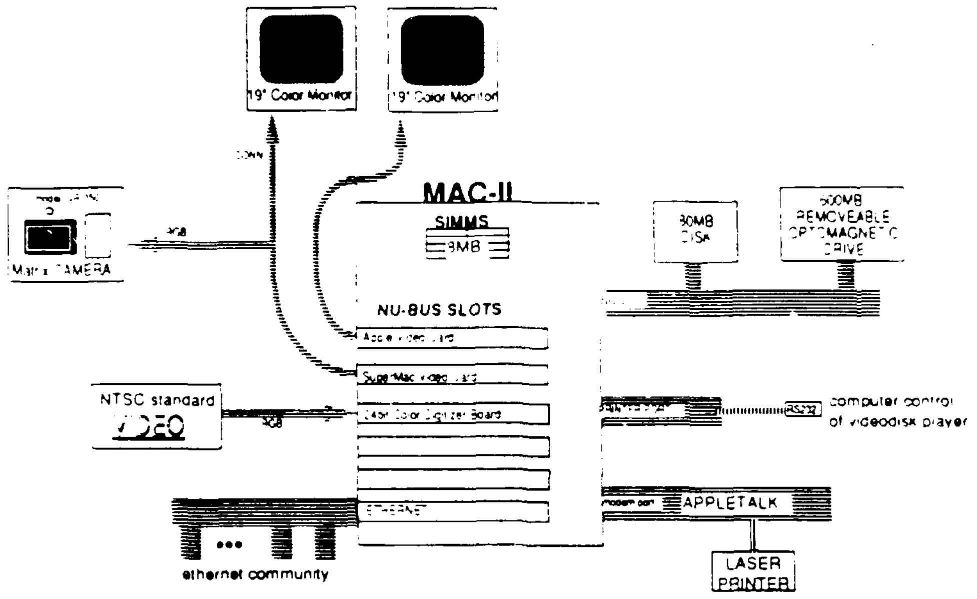


Figure 7-2: Current Workstation Configuration

The current environment is shown in Figure 7-2. It includes a 600MB magneto optical disk with removable media, an additional hi-resolution color monitor and video card, hi-resolution hardcopy capabilities, and extensive video image acquisition and digitizing facilities (see Figure 7-3). Coprocessors have also been explored as a means to expand the computational power of the platform (e.g., we evaluated the TI MicroExplorer and Symbolics MacIvory LISP coprocessor boards). These modular extensions have provided us with a powerful and flexible environment for operating within the vision and image processing domains.

In addition to the base level system, we have added a RasterOps 19" color monitor (model 1948S) which retails for about $5K and a SuperMac Spectrum/8 video card (1024 x 768 x 8 bits) which retails for about $1500.

A SCSI connected Jasmin 600MB magneto optical removable drive was added (it uses the new Ricoh drive). Each cartridge holds 300MB on each side, it has an average access time of 50ms (about twice the time required for most magnetic drives, though writing on the optical media takes longer), and a 1.2MB data transfer rate. The drive retails for about $5K and each removable cartridge retails for about $270. For tape backups, we transfer data over the ethernet to our timeshare machines which have attached 9-track and 8mm tape drives.

Figure 7-3 shows the video image acquisition subsystem we have added to the base level system. NTSC video sources are provided by a Sony LDP-1500 videodisk player, 3/4" videotape (Sony VO5800 recorder, VO5850 Recorder, and RM440 remote editing deck), and a Sony color broadcast quality camera. A FOR-A digital

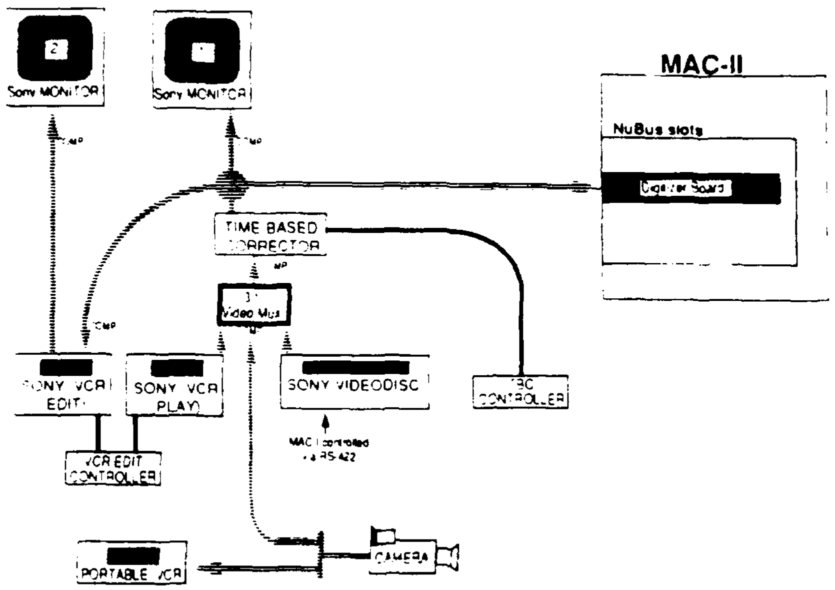


Figure 7-3: Video Image Acquisition Subsystem

time-based corrector reduces image noise by providing better video stability and freeze-frame capability (an especially important consideration for videotape). The videotape editing deck provides a desktop videotape controller which is placed near the operator console to allow convenient videotape and image access control. Because these decks do not have good freeze frame and random access abilities, they are not well-suited for the digitization of imagery in a motion sequence. The videodisk player and direct camera input has proven to be better for motion sequence digitization. The videodisk player is controlled by the Mac through its RS-422/232 port using the Voyager Videostack by The Voyager Company, Santa Monica, CA (about $60) which allows simple computer control of playback which includes forward play, fast play, single step, random frame jumps, freeze frame, rapid scan, disk eject and close, frame display, and unit reset. We plan to build a simple port interface to control the videodisk player via user program control and from within CommonLISP. Video data is displayed on two Sony video monitors.

We currently use an 8-bit monochrome Data Translation (DT) video digitizer board with four software-selectable input video channels (which we use as composite color, red, green, and blue) which retails for about $1100. The digitizer captures 640 x 480 images in 1/30th second. Color images may be captured by digitizing and saving the color planes in sequence. We currently drive the digitizer from the interactive program supplied by DT, and we also have a Pascal/C driver which allows direct capture of images from the digitizer into our preferred CVL image format. We eventually plan to integrate the current program interface to allow users to drive the digitizer functions from CommonLISP.

7.1.1 Future Mac-Based Systems

Several changes are currently underway in the Mac and general computing environment which will greatly impact the modular extensions available for the base level system.

Storage and archival technologies are rapidly developing faster, larger, and less expensive memories. The availability of large, low-cost secondary memories provides greater latitude in the computation/power tradeoff for IU environments, research, and applications development. Magneto optical media and 8mm tape systems will most rapidly develop in the near future. The recent advent of DMA boards has overcome some limitations in the Mac II systems (e.g., slow SCSI chip set, no DMA) which will result in far lower average access times for these secondary devices.

Coprocessors for the Mac and parallel processing in general is rapidly evolving. More parallel coprocessors are becoming available for the Mac, and the use of these coprocessors allows the overall computational power of the system to be stated in terms of hundreds of MIPS (versus the 3MIPS in the Mac x-series machines). In this framework, the Mac is primarily a chassis and graphics front-end processor. The recent development by Apple of MR-DOS is intended to standardize the communication and tasking in such an arrangement, and this standard may help to accelerate the development of these systems. The rumored addition by Apple of an AMD 29000 RISC graphics coprocessor to the Mac (supposedly to be introduced in late '89 or early '90) will offload graphics tasks from the main Mac CPU and significantly boost overall graphics and processing power.

Image acquisition and display technologies are also rapidly developing. It appears likely that within the next year or so, real-time digitizer to large optical media storage will be commercially available. Such systems are also capable of scanning digital data to video in real-time. This would provide high quality, random access, digital image acquisition, and real-time display of processing results at relatively low cost; current hard disk systems which currently do this run into the six digits. The ability to redisplay processing results in real-time which may have taken far longer to process (e.g., motion processing or segmentation results) will greatly help to describe algorithms and results in the community.

The basic sensory modalities of the user interface include visual (via the display), touch (via the keyboard and mouse), and sound (via the built-in Mac stereo sound and voice generation capabilities). Other sensory modalities may be added using off-the-shelf components to expand the ease, power, and naturalness with which the user may interact with the IU environment. For example, the interface may be expanded to include voice keyword recognition (e.g., the VoiceNavigator by Articulate Systems of Berkeley, CA retails for about $750 and operates without application software modification), a data glove (soon to be available by Nintendo at relatively low cost), 6-d trackball (e.g., from CIS Graphics, Westford, MA for $3300 retail), a touch screen (e.g., from MicroTouch Systems, Woburn, MA retail for $900 and up), and other sensory input devices. These additional sensory modalities expand the potential application domains, ease of use, and user expressiveness over the base level system.

In general, we found the Mac to be a very good graphical platform which facilitated the rapid prototype development of an IU environment on a low-cost, modularly expandable system. The base Mac system without the addition of coprocessor boards did not have sufficient computational power to support moderate to high computational requirements which usually occur in advanced IU applications. The use of coprocessor boards was explored and found to be a viable way to greatly increase the power of the base system to better handle increased computational loads. Upcoming improvements to the Mac operating system to handle virtual memory, multiprocessing, and larger memories will make the Mac better suited for IU environments and applications.

7.2 Interface Software

The interface software we developed was based upon a small set of objects which were implemented using the interface objects in the ObjectLISP extensions to Coral CommonLISP [Coral Software Corp. - 1987]. These objects are based, in turn, on the basic interface components in the Mac Tool Box [Apple Computer - 88]. These are reviewed below.

7.2.1 Windows

Images and objects are viewed in windows. Objects are two dimensional entities such as points, curves, regions, and assemblages of these. When a three dimensional object is displayed in a window, it is explicitly decomposed into these two dimensional objects. Viewing operations such as panning and zooming can be applied to windows. In addition, windows can be linked so that the display in one window can be seen in another window with an arbitrary transformation associated with the link. Thus, a window can zoom onto another window to view something at increased resolution. A transformation such as multilevel thresholding can be associated with the link between windows.

7.2.2 Browsers

Browsers are used to interact with text-based displays of objects. In one form, the attributes and values of a single object are displayed in a browse table. In the multi-column form, the attributes of multiple objects are displayed in a single browse table. The value in a browse table can be changed directly by selecting the indicated cells and typing in a new value. Data Base operations can also be performed over browse tables. It is possible to interactively perform boolean queries over the entities in a browse table. As with windows, browse tables can also be linked together so that changes in one can be propagate to another, as in changing the transformation matrix in one browse table to effect the position of coordinates of an object stored in another browse table associated with a spatial object.

7.2.3 Overlay Object Display

There are classes of objects corresponding to points, curves, regions, and assemblages of these which can be displayed on top of imagery. This is done for the projection of 3D graphical objects using these primitive display objects.

7.2.4 Label Planes

Associated with each image is a label plane (see Sections 2 and 5). This enables an object displayed in register with an image to have access to the underlying intensity values in the image and also to adjacent objects. Spatial queries can be applied to the object in the label plane.

7.3 Processing Examples

Figure 7-4 shows the appearance of the general set-up of the Mac II environment described in Section 7-1 while it is being used.

Figure 7-5 shows setting up an initial display window and associating a camera object with the world model and mousing through several different system options.

Figure 7-6 shows an image display window and a text browser window. The image display window contains a view onto the back parking lot at ADS. The user is selecting items (shown in yellow) from the text browser window from a library of images.

Figure 7-7 shows a display window and an associated text browser which describe the attachment relation between the camera object and the world model coordinate system.

Figure 7-8 shows three image display windows each containing a different image obtained from a 360 degree panoramic sequence of images obtained from the ADS parking lot.

Figure 7-9 shows multiple camera coordinate text browser associated with each image. The camera parameters can be altered interactively .

Figure 7-4: General Set-up of the Mac II Environment

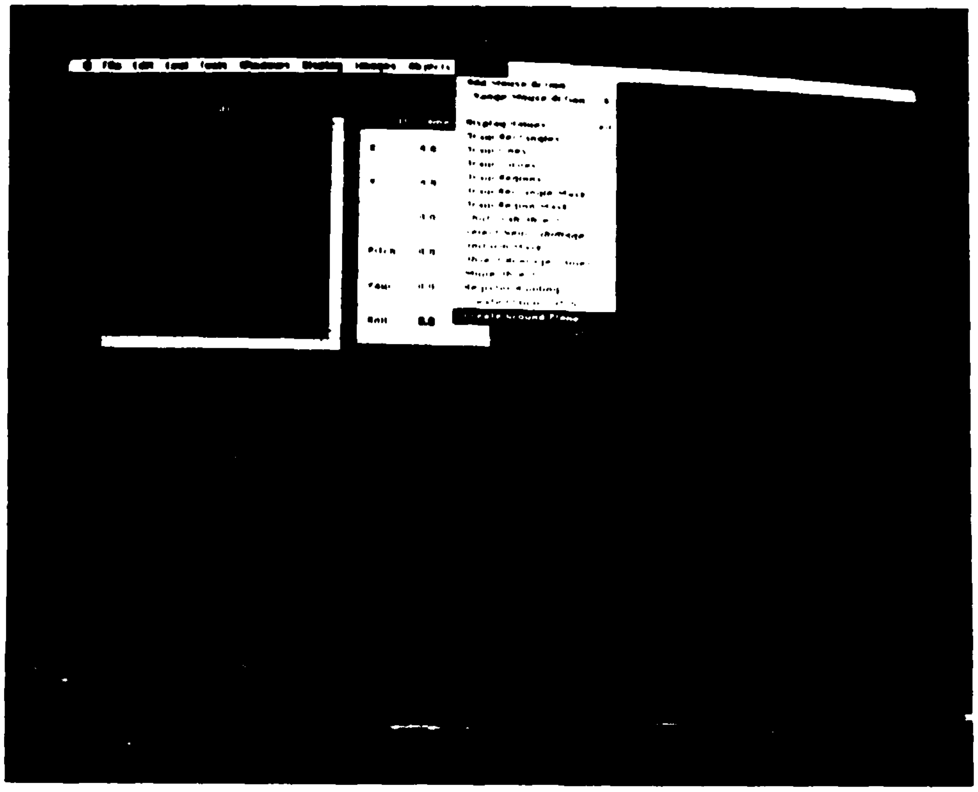


Figure 7-5: Display Window with Mouse-sensitive System Options

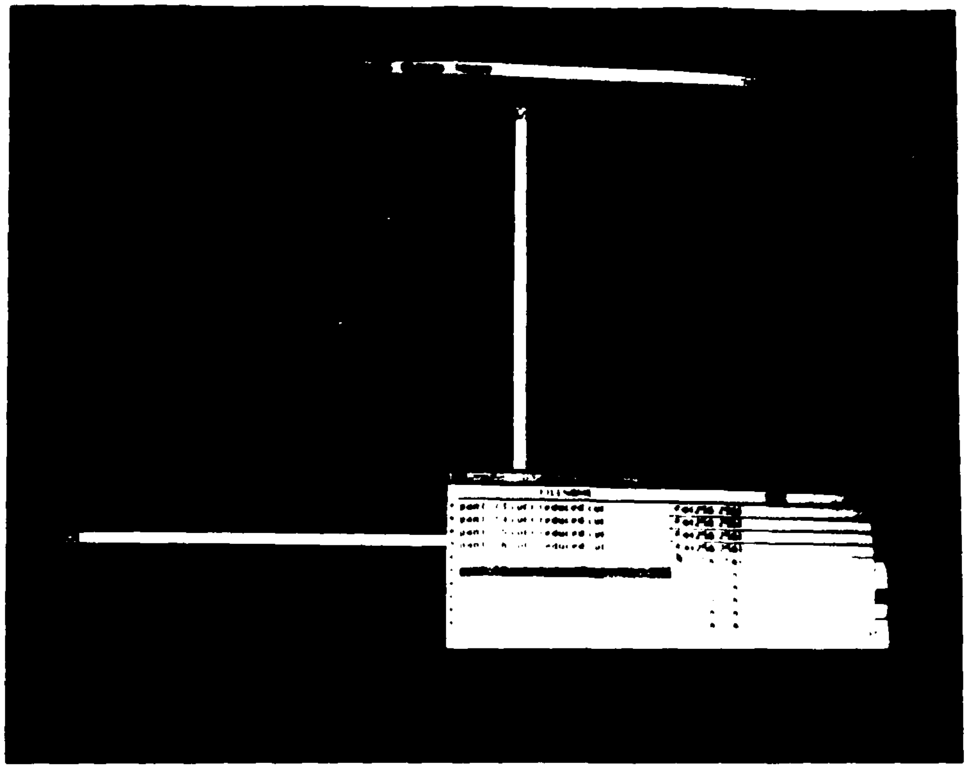


Figure 7-6: Image Display Window

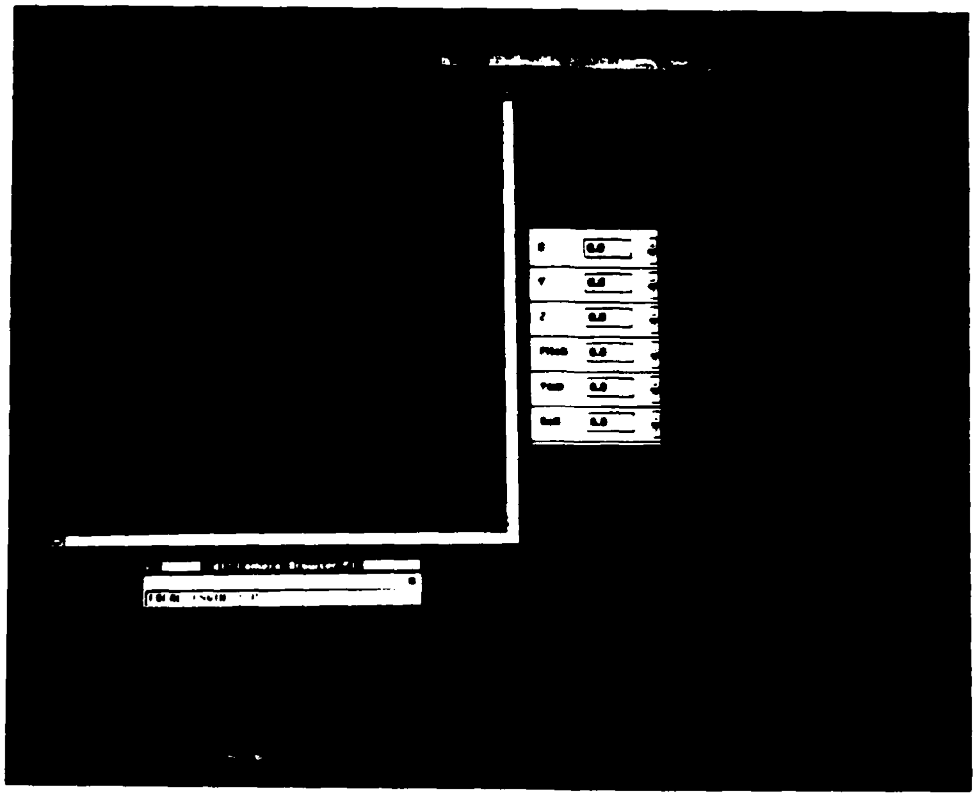


Figure 7-7: Display Window and an Associated Text Browser

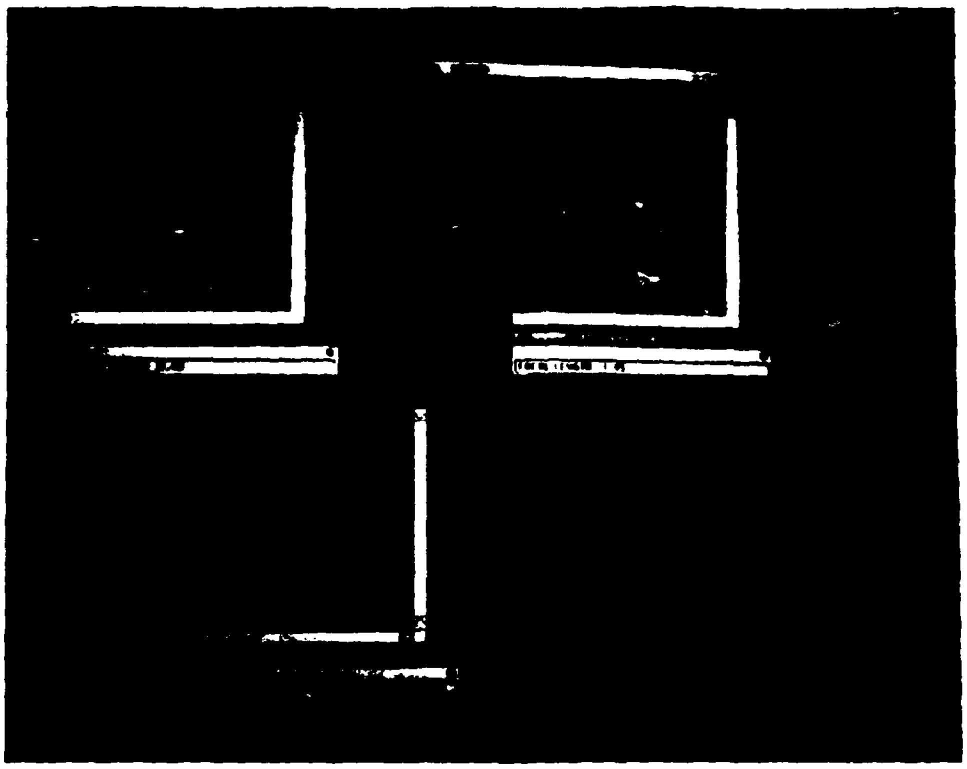


Figure 7-8: Three Image Display Windows

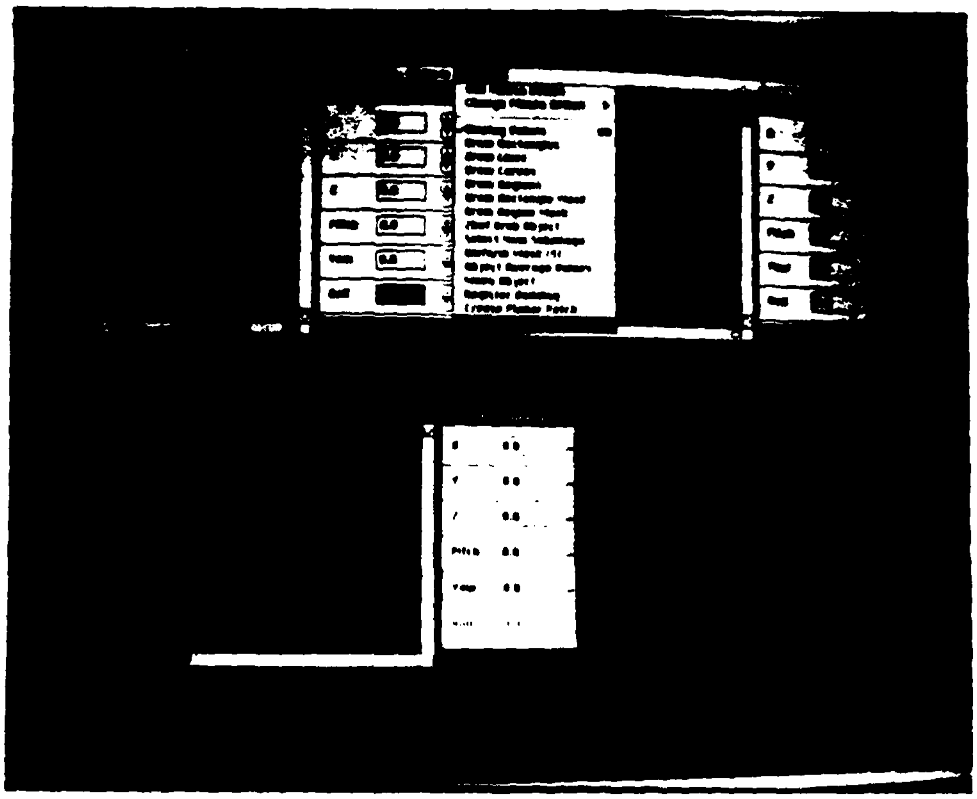


Figure 7-9: Multiple Camera Coordinate Text Browser

Figure 7-10: Multiple Linked Display Windows

Figure 7-10 shows multiple display windows linked by a *transformation link*. The transformation corresponds to a selecting and zooming on a subpart of the larger window. The displayed text browser window describes the parameters of the zoom and selection transformation between the windows.

Figure 7-11 shows the human interactively segmenting the image into curves that are deposited into the Perceptual Structure Data Base and can then be accessed via the label plane associated with the corresponding image. The corresponding data base of curves is displayed in a text browser window. The human can select and manipulate curves either via the label plane or by operations on the text browser window.

Figure 7-12 shows interactively placing an immediate ground plane object into the world model and back projecting it onto the image in the display window. The associated text browsers show the parameters describing the camera and the ground plane and their attachment.

Figure 7-13 shows the human using the mouse to access information about a displayed surface. The user can access the label plane associated with the displayed image. This allows access to the objects in the corresponding portions of the world model. In this case the attributes of the surface are displayed in a text browser window.

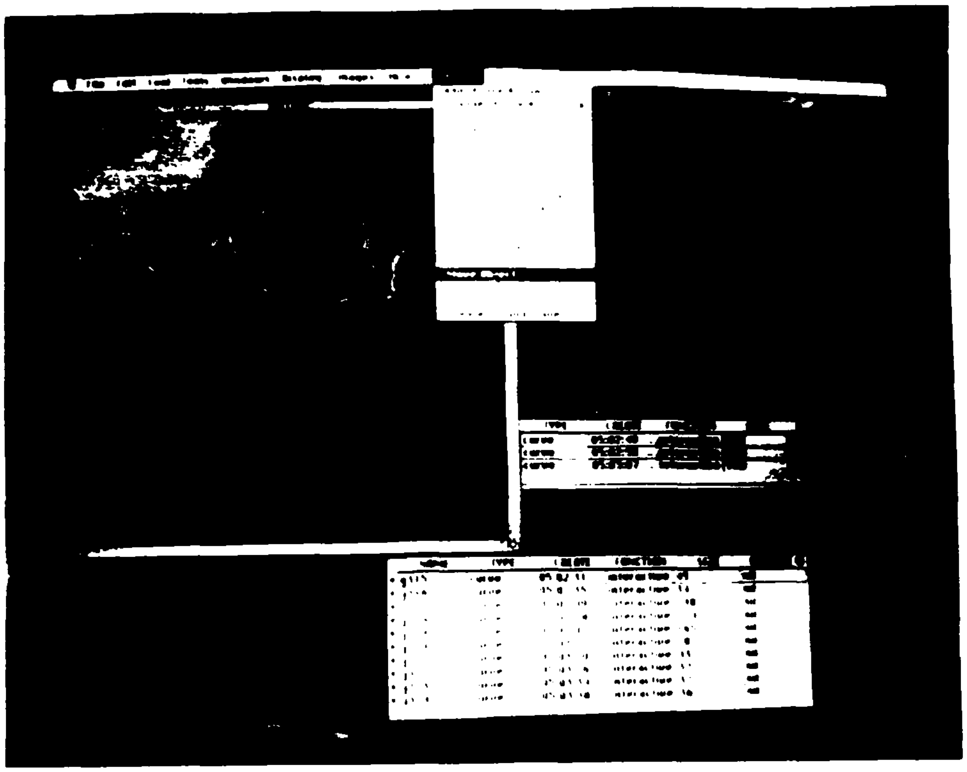


Figure 7-11: Human Interactively Segmenting the Image

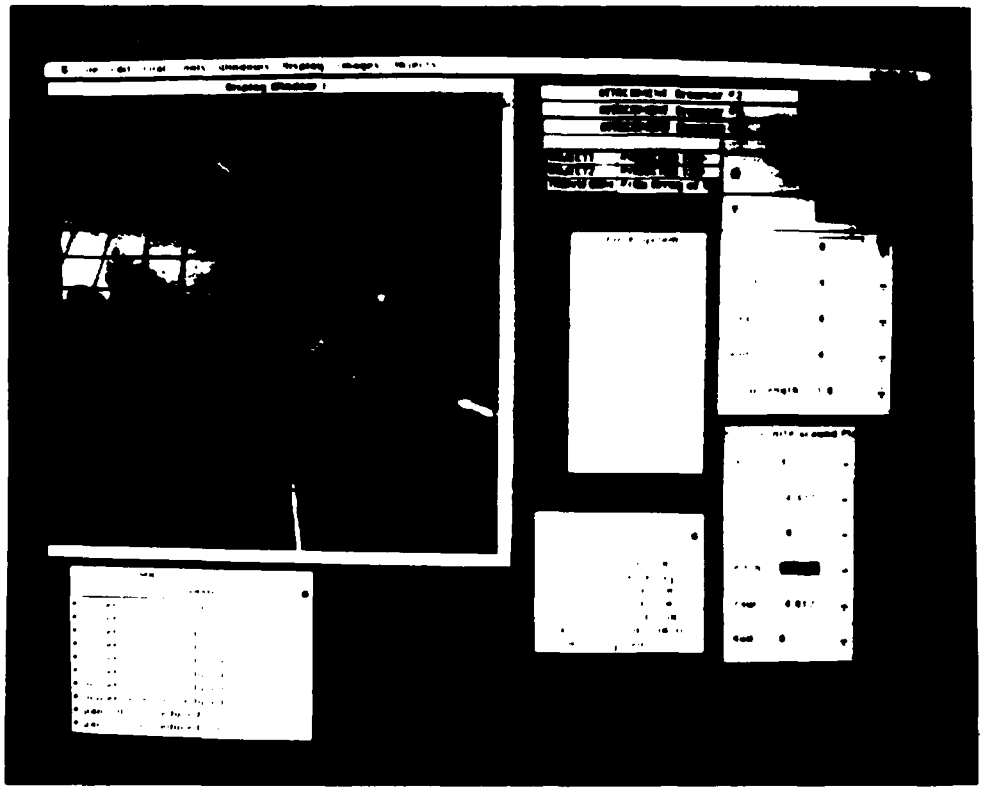


Figure 7-12: Placing Immediate Ground Plane Object

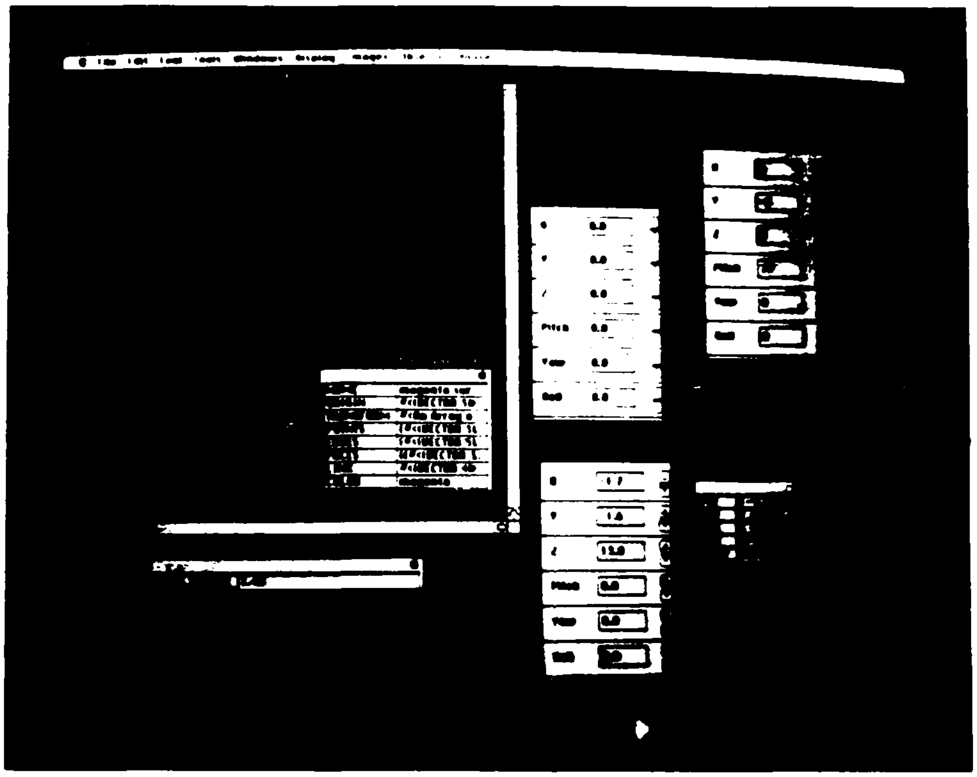


Figure 7-13: Human using the Mouse on Label Plane

8. Conclusions and Future Work

The objective of the first phase of this project has been to determine the feasibility and the design of an interactive model based vision system which will enable a human to rapidly interpret sensory data from a distributed team of telerobots. In Section 1 we described how such a system will enable a human to control a team of telerobots under low bandwidth communications. In Section 2 we presented the overall architecture of the system.

Section 3 reviewed work in constraint based reasoning and why it is necessary. The use of constraint based representations and reasoning has many advantages and, based upon alternatives we considered, is fundamental to realizing the system. Computational implementation of constraint based processing is the fundamental technical task for Phase II work, especially for working with complex object models. The difficulty of this will be simplified by the extent to which perceptual processing can obtain precise environmental information and that the human can intervene in the constraint satisfaction process with critical information.

Section 4 described our format for representing object models and it's implications for constraint-based reasoning and perceptual processing. We implemented simple object models for planar surfaces. A more extensive set of such objects will be required in Phase II work.

Section 5 described work in the representation and processing of perceptual information. This work has shown the ability to extract patterns that will correspond to the predictions generated by instantiated object models. Further work is required in simplifying the specification of a grouping process as an automatic part of object model instantiation. In addition, the library of passive processing routines for obtaining environmental information needs to be implemented.

Section 6 described work in the representation of large scale space and generation of visual predictions from a prior terrain information and previous human directed interpretations. This could be an effective initial demonstration of the system.

Section 7 reviewed our work in developing an initial version of the interactive interface for the system on a Mac II. The Mac II will be useful as an interface tool integrating future system components for such things as voice and gestural input. The Mac II has certain limitation so a workstation with a conventional operating system, such as a SUN, will also be used for Phase II work. It will also be of use to interface to an established CAD package such as BRL [SECAD/VLD Computing Consortium - 1988].

Based upon our work to date, we feel that the system is feasible. The key technical questions to be addressed during Phase II work are:

Constraint Satisfaction System: Constraint based reasoning is fundamental to the system's representation of objects and the control of inference and consistency determination in the model of the world that the human creates interactively. It allows for the modular development of object models and minimizes the amount of information a human needs to specify to place an object into the model of the world. The effort will be implementing the underlying constraint mechanisms needed to represent objects and also implementing the constraint controller. This task corresponds to the Constraint Controller and the World Model Data Base in the system architecture.

Object Model Representation: This involves implementing the basic representation for describing objects. It needs to be general and extendable to deal with the wide range of situations the telerobot could be in. We will also develop several specific object models for use in demonstrations. This work corresponds to the Object Model Data Base in the system Architecture.

Passive Perceptual Processing: A concealed telerobot will require passive sensor processing techniques for extracting three-dimensional scene information and for instantiating object models. Passive techniques are essential for avoiding detection. The effort will determine the extent to which they could achieve the functionality of active sensors. This corresponds to portions of the perceptual processing controller and portions of the eventual processing on-board the Telerobot.

Demonstration Preparations: An important aspect of this work is defining and performing a demonstration of the key technical components in an integrated scenario. Efforts here will focus on defining an important and achievable demonstration. This will assure alignment with HEL objectives and address system integration issues.

There are a few portions of the system architecture which will not require emphasis during Phase II development. The Long Term Data Base is in many senses dependent on completion of work on building effective world models using our defined object models. It is also being addressed by several other efforts in terrain modeling and by previous ADS work in the Knowledge Based Vision Project as part of the Autonomous Land Vehicle Effort [Lawton et al. - 89, Levitt and Lawton - to appear, Lawton and Levitt - 89]. We are not stressing work in user interfaces and machine independent graphical packages and interfaces due to the on-going dramatic developments in these areas in the commercial sector. We intend to use in-house or commercially available graphics packages. The components of the user interface will become common and inexpensive in the immediate future. Toy companies are starting to make hand position sensors for less than $100. It is essential, however, to be aware of these developments and remain compatible with them.

Bibliography

[Aloimonos and Swain - 85] J. Aloimonos and M. Swain, "Shape from Texture", in Proceedings *International Conference on Artificial Intelligence*, pp. 926-931, 1985.

[Apple Computer - 88] Apple Computer, *Inside Macintosh*, Vols. I-V, Addison-Wesley Publ., Reading, Massachusetts, 1988.

[Bobrow and Winograd - 77] D. Bobrow and T. Winograd, "An Overview of KRL, a Knowledge Representation Language", *Cognitive Sci. 1*, Vol. 1, No. 1, January 1977. pp. 3-46.

[Bobrow et al. - 87] D. G. Bobrow, L. G. DeMichiel, R. P. Gabriel, S. Keene, G. Kiczales, and D. A. Moon, **Common Lisp Object System Specification**, 1987.

[Borning - 81] A. Borning, "The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory", *ACM Transactions on Programming Languages and Systems*, Vol. 3, No. 4, 1981. pp. 353ff.

[Borning - 79] A. Borning, "ThingLab, a Constraint-Oriented Simulation Laboratory", Ph.D. Thesis, Dept of CS, Stanford University, 1979.

[Brooks, Flynn and Marill - 88] R. A. Brooks, A. M. Flynn, and T. Marill, "Self Calibration of Motion and Stereo Vision", *Proceedings of the DARPA Image Understanding Workshop*, Cambridge, Massachusetts, April 1988, pp. 398-410.

[Canny - 83] J. F. Canny, "Finding Edges and Lines in Images", Technical Report, Artificial Intelligence Laboratory, June 1983.

[Castleman - 79] K. R. Castleman, **Digital Image Processing**, Prentice-Hall, Inc., 1979.

[Clark, Covington, and Whelan - 84] J. Clark, T. Covington, and W. Whelan, "An Introduction to Ninja", Rand Corp, Santa Monica, CA, 1984.

[Coral Software Corp. - 1987] "Allegro Common Lisp for the Macintosh", Coral Software Corp. and Franz, Inc., 1987.

[Cotterill - 85] R. Cotterill, **The Cambridge Guide to the Material World**, Cambridge University Press, 1985.

[Cox - 84] B. J. Cox, **Message/Object Programming: An Evolutionary Change in Programming Technology**, IEEE Software, January 1984.

[Dahl and Nygaard - 66] O. -J. Dahl and K. Nygaard, "SIMULA–An ALGOL-Based Simulation Language", *Commun. ACM*, Vol. 9, No. 9, September 1966. pp. 671-678.

[Doyle - 79] J. Doyle, "A Truth Maintenance System", *Artificial Intelligence*, Vol. 12, 1979. pp. 231ff.

[Edelson, et.al. - 88] D. Edelson, J. Dye, T. Esselman, M. Black, and C. McConnell. "VIEW Programmer's Manual", Advanced Decision Systems, Mountain View. California, June, 1988.

[Elcock et al. - 71] E. W. Elcock, J. M. Foster, P. M. D. Gray, J. J. McGregor, and A. M. Murray, "ABSET, a Programming Language based on Sets: Motivation and Examples", **Machine Intelligence**, B. Meltzer and D. Michie, eds., Edinburgh University Press, Edinburgh, Scotland, 1971. pp. 467-492.

[Foley and Van Dam - 82] J. D. Foley and A. Van Dam, **Fundamentals of Interactive Computer Graphics**, Addison-Wesley Publishing Company, 1982.

[Gelband and Lawton - 88] P. Gelband and D. Lawton, "Perceptual Grouping Network Architecture", *Conference of the International Neural Network Society*, Boston, Massachusetts, September 1988.

[Gibson - 86] J. Gibson, "The Ecological Approach to Visual Perception", Lawrence Erlbaum Associates, Publishers, Hillsdale, NJ, 1986.

[Hall et.al - 82] E. L. Hall, J. B. K. Tio, C. A. McPherson, and F. A. Sadjadi, "Measuring Curved Surfaces for Robot Vision", *Computer*, Vol. 15, No. 12, pp. 42-54.

[Hewitt - 77] C. Hewitt, "Viewing Control Structures as Patterns of Passing Messages", *Artificial Intelligence*, Vol. 3, No. 8, June 1977. pp. 323-364.

[Hillis - 85] W. D. Hillis, **The Connection Machine**, The MIT Press, Cambridge, Massachusetts, 1985.

[Kearney, Yang, and Zhang - 89] J. K. Kearney, X. Yang, and S. Zhang, "Camera Calibration using Geometric Constraints", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, San Diego, CA, June 1989, pp. 672-677.

[Lawton - 80] D. T. Lawton, "Constraint-Based Inference from Image Motion", *Proceedings of the First American Association of Artificial Intelligence*, Stanford, CA, August 1980.

[Lawton and Levitt - 89] D. T. Lawton and T. S. Levitt, "Knowledge Based Vision for Terrestrial Robots", *Proceedings of the DARPA Image Understanding Workshop*, Palo Alto, California, May 1989.

[Lawton and McConnell - 88] D. Lawton and C. McConnell, "Image Understanding Environments", in *IEEE Proceedings*, August, 1988.

[Lawton and McConnell - 87] D. T. Lawton and C. C. McConnell, "Perceptual Organization Using Interestingness", *AAAI Spatial Reasoning Workshop*, St. Charles, Illinois, October 5-7, 1987.

[Lawton, Rieger, and Steenstrup - 87] , D. T. Lawton, J. H. Rieger and M. S. Steenstrup, "Computational Techniques in Motion Processing", **Vision, Brain and Cooperative Computation**, MIT Press, M. Arbib and A. Hanson, eds., 1987, pp. 419-488.

[Lawton et al. - 89] D. T. Lawton et al., "Knowledge-Based Vision Techniques Task B: Terrain and Object Modeling Recognition", TR-1093-04, Advanced Decision Systems, Mountain View, California, 1989.

[Lawton et.al. - 87] D. T. Lawton, T. S. Levitt, C. C. McConnell, and P. C. Nelson, "Environmental Modeling and Recognition for an Autonomous Land Vehicle", *NASA Telerobotics Conference*, Pasadena, California, January 20-23, 1987.

[Leler - 88] Wm. Leler, **Constraint Programming Languages: Their Specification and Generation**, Addison-Wesley, Reading, MA, 1988.

[Lenz and Tsai - 88] , R. Lenz and R. Y. Tsai, "Calibrating a Cartesian Robot with Eye-on-Hand Configuration Independent of Eye-to-Hand Relationship", *IEEE Computer Vision and Pattern Recognition*, Ann Arbor, MI, June 1988. Also in *IEEE Transactions on PAMI.*

[Levitt and Lawton - to appear] T. S. Levitt and D. T. Lawton, "Qualitative Navigation for Mobile Robots", *AI Journal*, to appear.

[Lieberman and Hewitt - 80] H. Lieberman and C. Hewitt, "A Session with TINKER: Interleaving Program Testing with Program Design", *1980 Lisp Conference*, Stanford, CA, August 1980. pp. 90-99.

[Liskov et al. - 77] B. Liskov, A. Snyder, R. Atkinson, and C. Shaffert, "Abstraction Mechanisms in CLU", *Communication of the ACM*, Vol. 8, No. 20, August 1977. pp. 564-576.

[Luh and Klaasen - 85] J. Y. S. Luh, and J. A. Klaasen, "A Three-Dimensional Vision by Off-Shelf System with Multi-Cameras", PAMI-7,1,(Jan.) 35-45.

[Martelli - 76] A. Martelli, "An Application of Heuristic Search Methods to Edge and Contour Detection", *Commun. ACM*, Vol. 19, No. 2, February 1976, pp. 73-83.

[McAllester - 78] D. A. McAllester, "A Three-Valued Truth Maintenance System", MIT AI Lab Memo 473, 1978.

[McConnell et.al. - 87] C. McConnell, P. Nelson and D. Lawton, "Constructs for Co-operative Image Understanding Environments", *Proceedings of the DARPA Image Understanding Workshop*, Los Angeles, California, February, 1987.

[Merritt - 88] John O. Merritt, "Virtual Window Viewing Geometry", *Sensor Fusion: Spatial Reasoning and Scene Interpretation*, Cambridge, MA, November 1988, pp. 386-389.

[Mitchell, Maybury, and Sweet - 79] J. Mitchell, W. Maybury, and R. Sweet, "Mesa Language Manual", CSL-79-3, Xerox PARC, Palo Alto, CA, April 1979.

[Mundy, Vrobel and Joynson - 89] J. L. Mundy, P. Vrobel, and R. Joynson, "Constraint-Based Modeling", GE Corporate R&D Center, 1989.

[O'Reilly & Associates, Inc. - 1988] O'Reilly & Associates, Inc., **Xlib Reference Manual**, Volumes I-III, Adrian Nye, Editor, 1988.

[Peterson] Larry Peterson, personal communication.

[Riley et.al. - 87] K. Riley, C. McConnell, and D. Lawton, "Powervision", Advanced Decision Systems, Mountain View, California, April, 1987.

[Ruoff et al. - 84] C. Ruoff, J. Bowyer, T. Brooks, J. Hanson, K. Holmes, and B. Wilcox, "Autonomous Ground Vehicles: Control System Technology Development", Jet Propulsion Laboratory, Pasadena, CA, October 1984.

[Schmucker - 86] K. J. Schmucker, **Object-Oriented Programming for the Macintosh**, Hayden Book Company, 1986.

[Schwartz - 86] A. Schwartz, "Head Tracking Stereoscopic Display", *Society for Information Display*, 1986, pp. 133-137.

[SECAD/VLD Computing Consortium - 1988] "The Ballistic Research Laboratory CAD Package: A Solid Modeling System and Ray-Tracing Benchmark", SECAD/VLD Computing Consortium, 1988.

[Smith - 75] D. Smith, "PYGMALION: A Creative Programming Environment", AIM-260, Dept. Computer Science, Stanford University, June 1975.

[Stallman and Sussman - 78] R. M. Stallman and G. J. Sussman, "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis", *Artificial Intelligence*, Vol. 9, 1978. pp. 135ff.

[Steeb et al. - 86] R. Steeb, S. Cammarata, S. Narain, J. Rothenberg, and W. Giuarla, "Cooperative Intelligence for Remotely Piloted Vehicle Fleet Control, Analysis, and Simulation", Rand Corp, Santa Monica, CA, 1986.

[Steele, Jr. - 80] G. Steele, Jr., "The Definition and Implementation of a Computer Programming Language based on Constraints", Ph.D. Thesis, Dept. Electrical Engineering and Computer Science, M.I.T., Cambridge, MA, August 1980. Available as MIT-AI TR 595, August 1980.

[Steele, Jr. and Sussman - 78] G. L. Steele, Jr. and G. J. Sussman, "Constraints", MIT AI Lab Memo 502, M.I.T., Cambridge, MA, November 1978. Also in APL '79: Conference Proceedings, *APL Quote Quad* (ACM SIGPLAN/STAPL) 9, 4 (June 1979), part 1, pp. 208-225.

[Steels - 79] L. Steels, "Reasoning Modelled as a Society of Communicating Experts", MIT-AI TR 542, M.I.T., Cambridge, MA, 1979.

[Stefik - 81] M. J. Stefik, "MOLGEN part 1: Planning with Constraints", *Artificial Intelligence*, Vol. 2, No. 16, May 1981. pp. 111-139

[Sutherland - 63] I. Sutherland, "Sketchpad: A Man-Machine Graphical Communication System", Ph.D. Thesis, Dept. Electrical Engineering, M.I.T., Cambridge, MA, 1963. Also appears in *Proceedings of the IFIP Spring Joint Computer Conference*, 1963.

[Tsai - 87] R. Tsai, "A Versatile Camera Calibration Technique for High Accuracy 3D Machine Vision Metrology using Off-the-shelf TV Cameras and Lenses", *IEEE Journal of Robotics and Automation*, Vol. RA-3, No. 4, August 1987. A preliminary version appeared in *1986 IEEE International Conference on Computer Vision and Pattern Recognition*, Miami, Florida, June 22-26.

[Tsai and Lenz - 88a] Roger Y. Tsai and Reimar K. Lenz, "Overview of a Unified Calibration Trio for Robot Eye, Eye-to-Hand, and Hand Calibration using 3D Machine Vision", *Sensor Fusion: Spatial Reasoning and Scene Interpretation*, Cambridge, MA, November 1988, pp. 202-213.

[Tsai and Lenz - 88b] R. Y. Tsai and R. Lenz, "Real Time Versatile Robotics Hand/Eye Calibration using 3D Machine Vision", *International Conference on Robotics and Automation*, Philadelphia, PA, April 1988. Also to appear in *IEEE Journal of Robotics and Automation*, Spring of 1989.

[Valyus - 66] N. A. Valyus, **Stereoscopy**, Focal Press, London & New York, 1966, pp. 377-385.

[Van Wyk - 80] C. J. Van Wyk, "A Language for Typesetting Graphics", Ph.D. Thesis, Dept of CS, Stanford University, 1980.

[Weems and Levitan - 87] C. C. Weems and S. P. Levitan, "The Image Understanding Architecture", in Proceedings *Image Understanding Workshop*, pp. 483-496, Los Angeles, California, February, 1987.

[Wolfram - 88] S. Wolfram, **Mathematica**, Addison-Wesley Publishing Company, 1988.

[Wulf, London, and Shaw - 76] W. A. Wulf, R. London, and M. Shaw, "An Introduction to the Construction and Verification of Alphard Programs", *IEEE Transactions on Software Engineering*, Vol. 4, No. SE-2, December 1976. pp. 253-264.